UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ALCIDES SILVEIRA COSTA

An emulation-based remote laboratory for prototyping digital circuits without FPGA

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dr. André Reis

Porto Alegre
June 2023

CIP — CATALOGING-IN-PUBLICATION

Costa, Alcides Silveira

An emulation-based remote laboratory for prototyping digital circuits without FPGA / Alcides Silveira Costa. – Porto Alegre: PPGC da UFRGS, 2023.

128 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2023. Advisor: André Reis.

1. Remote laboratory. 2. Circuit design. 3. Emulation. 4. Fpga. I. Reis, André. II. Título.

To Whom it revealed me
during the darkest days of my life
the meaning of sacrifice
through the love of my wife

ABSTRACT

This thesis presents a novel approach to remotely prototyping digital circuits without FPGA-based prototyping boards. Motivated by the cost of FPGA-based prototyping boards and the mobility restrictions caused by the COVID-19 pandemic, the thesis presents an emulation-based platform, Pitanga, aiming to reduce hardware costs while allowing students to prototype digital circuits without being physically present in the laboratories of the educational institution. Instead of a physical prototyping board, the student interacts with a lightweight graphical user interface containing a virtual prototyping board on the computer. The platform uses a client-server architecture that offloads the student's computer by running the design and the emulation software on the server-side. This approach is an alternative to FPGA-based remote laboratories. However, the Pitanga platform responds to the students' stimuli with near-zero latency and employs general-purpose CPUs to emulate the remote physical FPGA-based prototyping board. The Pitanga platform responds with reduced latency because it runs a predictive emulator on the server-side that calculates the output for every possible input state of the virtual prototyping board running on the client-side. The results show that the Pitanga platform can emulate a digital circuit of 72,486 transistors at 1Hz system clock. This complexity is the equivalent of an Intel 8086 implemented in NMOS technology or a 1024-bit counter implemented in CMOS technology. Also, the results show that the predictive emulator is $O(n)$ time.

Keywords: Remote laboratory. circuit design. emulation. fpga.

Laboratório remoto baseado em emulação para a prototipação de circuitos digitais sem FPGA

RESUMO

Esta tese apresenta uma abordagem inovadora para a prototipagem remota de circuitos digitais sem o uso de placas de prototipagem baseadas em FPGA. Motivado pelo custo dessas placas e pelas restrições de mobilidade causadas pela pandemia COVID-19, a tese apresenta a plataforma Pitanga, baseada em emulação, com o objetivo de reduzir os custos de hardware e permitir que os alunos criem protótipos de circuitos digitais sem estar fisicamente presentes nos laboratórios da instituição de ensino. Em vez de uma placa de prototipagem física, o aluno interage com uma interface gráfica de usuário, leve, contendo uma placa de prototipagem virtual no computador. A plataforma utiliza uma arquitetura cliente-servidor que executa o software de design e emulação no lado do servidor, portanto, diminuindo a carga computacional no computador do aluno. Essa abordagem é uma alternativa aos laboratórios remotos baseados em FPGA. A plataforma Pitanga responde com latência próxima de zero aos estímulos dos alunos utilizando CPUs de propósito geral para emular placas de prototipação baseada em FPGA. A redução de latência ocorre porque há um emulador preditivo no lado do servidor que calcula as respostas para todos os possíveis estados de entrada da placa de prototipagem virtual em execução no lado do cliente. Os resultados mostram que a plataforma Pitanga pode emular um circuito digital de 72.486 transistores a 1Hz de clock do sistema. Essa complexidade equivale a um Intel 8086 implementado em tecnologia NMOS ou a um contador de 1024 bits implementado em tecnologia CMOS. Além disso, os resultados mostram que o emulador preditivo possui complexidade $O(n)$.

Palavras-chave: Laboratório remoto, emulação de circuitos, educação em engenharia.

LIST OF ABBREVIATIONS AND ACRONYMS

AIG	And-Inverter Graph
ASIC	Application Specific Integrated Circuit
AST	Abstract Syntax Tree
CMOS	Complementary Metal-Oxide-Semiconductor
CTS	Clock Tree Synthesis
DAG	Directed Acyclic Graph
DFT	Design For Test
DUT	Design Under Test
EDA	Electrical Design Automation
GLS	Gate-Level Simulation
GUI	Graphical User Interface
ICE	In-Circuit Emulation
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
IC	Integrated Circuit
MCU	MicroController Unit
PCB	Printed Circuit Board
RTL	Register-Transfer Level
SaaS	Software-as-a-Service
SAIF	Switching Activity Interchange Format
SoC	System-on-Chip
VCD	Value Change Dump
VLSI	Very-Large Scale Integration

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Since March 2020, human society has been severely affected by the COVID-19 pandemic (CIOTTI et al., 2020). The pandemic has forced governments worldwide to adopt social distancing policies, even imposing lockdowns and face masks on people to avoid spreading the new virus. Companies suspended in-person activities, countries closed their borders, and the world supply chain has slowed down in response to such political measures (JAFFE, 2021; TIMUR; XIE, 2021; CAI; LUO, 2020). In the higher education sector, universities worldwide had to adapt and provide distance learning courses to keep the classes going (GAUDIOT; KASAHARA, 2020; LIEBOWITZ, 2020).

Distance learning impacted not only the delivery of traditional face-to-face classes but also the delivery of hands-on laboratory experiments. Hands-on laboratories require physical access to equipment, challenging IT university departments to provide the laboratory infrastructure through the Internet. Consequently, the number of publications related to virtual and remote laboratories has increased dramatically during this period to provide solutions to this problem (RAMAN et al., 2022; RODA-SEGARRA, 2021).

In electrical and computer engineering courses, hands-on laboratories usually employ FPGA-based prototyping boards for digital circuit design experiments. During COVID-19, many universities could not use their laboratory infrastructure (computers, FPGA boards, and related software) because in-person lab activities were not permitted. Although some universities have managed to share the laboratory infrastructure online for their students (MELOSIK et al., 2022; ALMEIDA et al., 2022), each university is unique, with different facilities, staff, budgets, and professors. Not every university can afford to buy FPGA-based prototyping boards for hands-on laboratories, much less finance a dedicated staff to support the infrastructure required to share the lab online. Therefore, a generation of students had the quality of their engineering education negatively affected by the lack of hands laboratories (ASGARI et al., 2021). This practical problem motivates the effort to develop more cost-efficient tools and teaching alternatives while still remotely providing students with the digital circuit design experience.

1.1 A glance at the FPGA remote laboratory

The effort to share lab infrastructure over the Internet is not new. The first records date back to 1996 with the implementation of a robot arm experiment controlled over the Internet at the College of Engineering of the Oregon State University (AKTAN et al., 1996). MIT also contributed to the microelectronics field by making device characterization over the Internet possible in 2003 with WebLab, delivering the laboratory experience to any user with a conventional web browser (FJELDLY; SHUR, 2005).

In the digital circuit design arena, the popularization and ever-growing capacity of FPGA devices (TRIMBERGER, 2015) contributed to replacing digital circuit laboratories using traditional breadboards and TTL components with FPGA-based prototyping boards (NICKELS, 2000; BOULDIN, 2004). This technological advancement made possible the emergence of web-based digital circuit design laboratories in the early 2000s (BECKER et al., 1998; IZUMI et al., 2001; MCCRACKEN; ZILIC; CHAN, 2003), starting a new research field known in the scientific literature as FPGA remote laboratory.

Although FPGA remote laboratories have been researched for more than two decades, this approach has two fundamental limitations that could not be solved entirely. They are:

1. FPGA remote laboratories are not fault-tolerant, requiring a very responsive maintenance service to fix and replace unexpected hardware failures and malfunctions;
2. FPGA remote laboratories rely on costly hardware acquisition for scaling, imposing additional cost challenges for anyone employing this solution.

These two aspects are the heart of the recent issues researched in the literature today. For this reason, to find a solution for the issue described in item 1, Villar-Martinez et al. presented a recent study about fault-detection techniques to automatically identify hardware malfunctions in FPGA remote laboratories (VILLAR-MARTINEZ et al., 2021). The authors' investigation did not solve the faults but attenuated the need for a responsive maintenance service. To obtain a solution for the lab scaling, as described in item 2, the same authors evaluated the effectiveness of FPGA remote laboratories replication (VILLAR-MARTINEZ et al., 2022). The

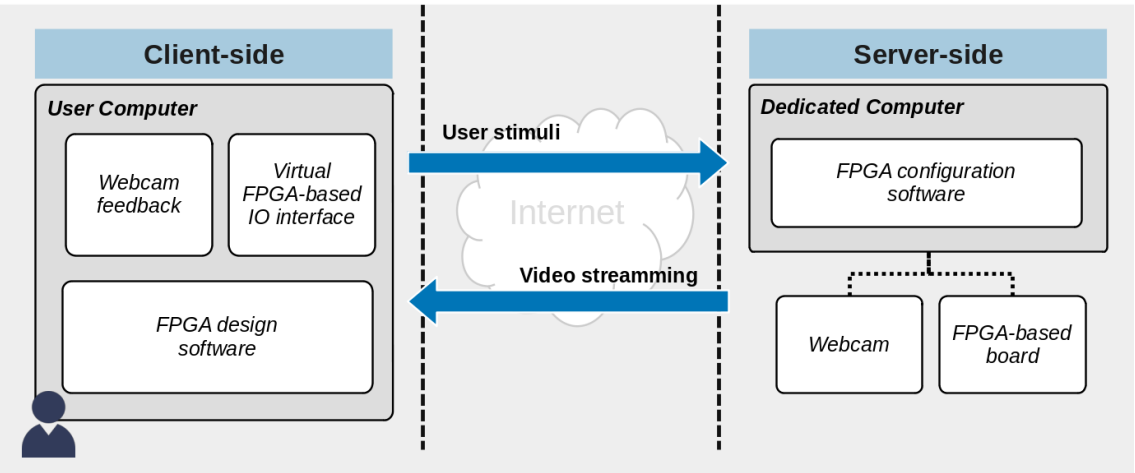
evaluation has considered educational institutions sharing FPGA-based boards in a network of several FPGA remote laboratories. In this case, the remote lab scales due to the addition of FPGA-based boards shared by other institutions in the same lab network.

In both cases, the authors did not present a definitive solution for automatic fault detection in the remote lab equipment, just as they did not show a definitive solution for laboratory scaling without the receipt of buying additional FPGA-based boards. Thus, both issues described in items 1 and 2 remain without a cost-affordable and fully-automated solution.

1.1.1 The FPGA remote laboratory architecture

FPGA remote laboratories have been evolving since their first appearance in the early 2000s. However, independently of the advancements, all FPGA remote laboratories have the same principle: an off-the-shelf FPGA-based prototyping board connected to a computer that shares this same board over the Internet. Figure 1.1 depicts the basic architecture of this type of lab.

Figure 1.1: Traditional FPGA remote laboratory overall architecture



Notice that the traditional architecture requires a computer, an FPGA-based board, and a webcam on the server-side. The computer acts as a proxy server, allowing users on the client-side to access the FPGA-based board. The webcam is usually positioned above the FPGA-based board to capture light stimuli - i.e., LEDs and seven-segment display changes - so the remote user can check whether his or her design is working correctly.

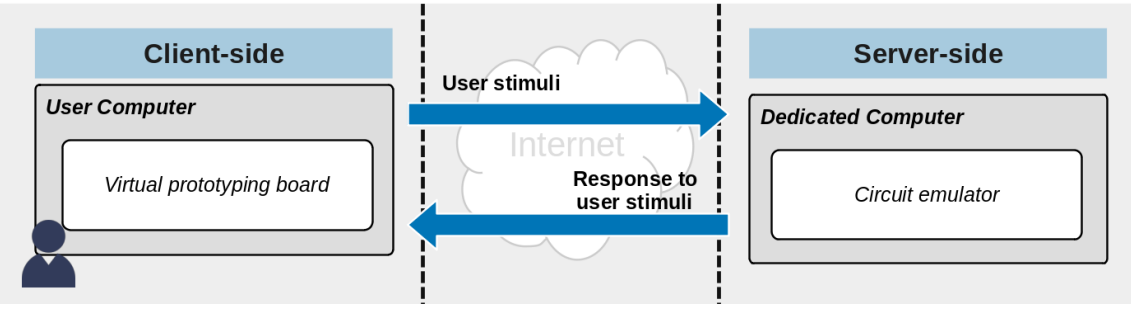
The traditional architecture has the disadvantage of demanding facilities (room, electricity, cooling), network and computing hardware (network equipment, computers, cables), and people to support the underlying IT software and services (operating system, design software, software updates). These demands naturally impose additional costs on the educational institution. For this reason, educational institutions with a limited budget commonly choose digital circuit simulators as an alternative to practical laboratories.

However, although the digital circuit simulator approach is less costly, simulators operate with previously defined input stimuli. The user must code the input stimuli, either in Verilog or VHDL format, run the simulator and check the results (either visually or in an automated way). There is no real-time interaction between the user and the designed circuit when the simulator runs. Indeed, simulators are not meant to provide a real-life experience: instead, simulators are meant to provide a debugging environment for verifying digital circuits. Thus, the sole use of simulators in digital circuit laboratory sessions eliminates the primary purpose of the lab: to provide a practical experience that the students will get when using real-life FPGA-based prototyping boards.

1.1.2 The proposed remote laboratory architecture

This thesis presents a cost-efficient alternative architecture - named Pitanga platform (COSTA; DROVES; REIS, 2023a) - for experimenting with digital circuits. Following the recent trends in Laboratory as a Service (LaaS) (RAMOS; ALBERTINI; SOLIS-LASTRA, 2022), this thesis proposes an emulation-based platform as an alternative for the traditional FPGA remote laboratories, as shown in Figure 1.2.

Figure 1.2: Pitanga emulation-based remote laboratory overall architecture



The proposed architecture differentiates from the traditional architecture be-

cause it does not provide physical FPGA-based prototyping boards. Instead, the system delivers a virtual prototyping board that runs on any general-purpose computer on the client-side. The virtual board captures the user stimuli and transmits to the server-side for further processing. The server-side emulates the digital circuit - previously built by the Pitanga compiler - and transmits the response back to the client-side. The server-side also runs on a general-purpose architecture.

As the Pitanga client-server architecture runs on a general-purpose computer, both sides scale with cost-affordable general-purpose computers. The proposed architecture also delivers digital circuit prototyping as a software service for digital circuit classes. This approach eliminates common hardware faults in FPGA remote laboratories (VILLAR-MARTINEZ et al., 2021) by implementing the FPGA-based board in software. Thus, the proposed platform addresses the two heart issues listed in Section 1.1.1: the need for a responsive maintenance team and the FPGA lab scaling cost.

Compared to the traditional FPGA remote laboratory in Figure 1.1, the Pitanga platform replaces the FPGA-based board on the server-side with a responsive digital circuit emulator. The emulator receives the user stimuli captured by the rendered virtual board on the client-side and produces a response to these inputs. The emulator sends back the responses to the client-side, which updates the output components of the virtual board (LEDs and seven-segment displays). So, this architecture replaces both the webcam and the FPGA-based board, reducing lab costs and enabling scaling.

The Pitanga architecture comes as a cost-affordable alternative approach to the practical problem of prototyping digital circuit designs remotely and without FPGA-based prototyping boards. However, the Pitanga architecture depicted in Figure 1.2 is unclear about its benefits as an educational technology. According to Froyd et al. (FROYD; WANKAT; SMITH, 2012), acceptance of educational technologies benefit from being inexpensive, easy to use, and based on standards extensively used in the industry. Considering the Pitanga platform is also a product that pursues acceptance as an educational technology, Table 1.1 lists some of the Pitanga platform requirements, each one associated with the acceptance criteria proposed by Froyd et al.:

Table 1.1: Pitanga platform requirements associated with acceptance criteria of educational technologies applied to digital circuit remote laboratories

No	Requirement	Criteria
1	Widely used commercial HDL as input	based on industry-standard
2	Similar design flow and design stats in comparison with commercial design tools (i.e., area, cells usage, power, and timing report)	based on industry-standard
3	Design software integration capability with commercial design tools	based on industry-standard
4	Use of the established institution computing hardware for immediate deployment of the remote laboratory	inexpensive
5	Low cost for scaling up and down the digital circuit lab	inexpensive
6	No need for a highly available and responsive lab maintenance staff	inexpensive
7	Virtual circuit board with look and feel similar to typical FPGA-based educational boards	easy to use
8	Non-confusing design software with a lightweight installation process	easy to use
9	Useful documentation with significant teaching material	easy to use
10	User stimuli response in real-time by the remote emulation software	undefined

From a product perspective, the Pitanga platform proposes to address all the requirements listed in the Table 1.1. However, from a research perspective, the tenth requirement demands special attention and raises the following questions:

- What maximum network latency must the proposed architecture deliver to be acceptable as a real-time emulator?
- How fast can an emulated digital circuit run on the proposed platform?
- How many logic cells can the proposed platform emulate?

The answer to these research questions resides in the goal of this thesis.

1.2 Thesis

This thesis claims that students can prototype digital circuits remotely and without physical FPGA-based boards. By using the proposed architecture described in Figure 1.2, this thesis implements a digital circuit emulation software and demonstrates that it is possible to prototype digital circuits as a software service with real-time response to user stimuli. With the help of EDA data structures extensively used in modern logic synthesis tools, the proposed system resembles the FPGA remote laboratory experience, with the advantage of bypassing the network latency inherent to the client-server architecture.

The main contribution of this thesis is the real-time response that the system delivers. In order to do that, the Pitanga platform employs techniques from networked games (GLAZER; MADHAV, 2015) and logic synthesis tools (WANG; CHANG; CHENG, 2009). The result is a digital circuit predictive emulator. In other words, the result of this thesis is an algorithm capable of predicting the digital circuit response to the user inputs before the user interacts, i.e., before the user even presses any of the push buttons and switches in the virtual prototyping board.

The iLabs from Stanford Univerity proposed a remote laboratory that can also predict responses to user stimuli (ZAMAN; NEUSTOCK; HESSELINK, 2021). Zaman et al. predict the responses by generating and storing an extensive data set of recorded laboratory experiments previously. The Zaman's proposal fits well for experiments with a finite number of states and experiments where network latency does not impose an issue. However, this is not the scenario of digital circuit experiments, in which non-responsive circuits may indicate a design error, and the number of circuit states is much more significant.

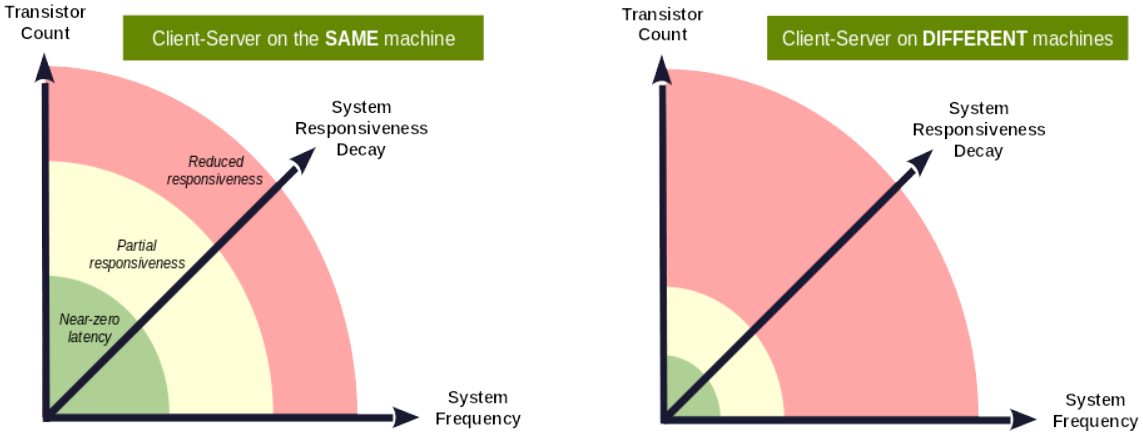
1.3 Goal

The goal of this thesis is to evaluate the effectiveness of the proposed client-server architecture in Figure 1.2 in responding to user stimuli. It aims to find the system responsiveness decay when the Pitanga emulation software runs remotely (i.e., virtual board and emulator on different machines) and locally (i.e., virtual board and emulator on the same machine), as depicted in Figure 1.3.

Figure 1.3 illustrates three distinct regions, each encompassing a range of

coordinates denoting $\{transistor\ count, system\ frequency\}$ pairs. The coordinates within the innermost semi-circle indicate the pairs of coordinates at which the Pitanga emulation system operates with near-zero latency, offering heightened responsiveness. On the other hand, the coordinates within the outermost semi-circle represent the coordinates where the Pitanga emulation system operates with latency that is perceptible to the user, resulting in reduced responsiveness. The region between the outer and inner areas denotes the pair of coordinates where the Pitanga system exhibits partial responsiveness, although not perceivable by the user.

Figure 1.3: Expected system responsiveness decay when the Pitanga emulation software runs remotely and locally



This thesis argues that the system responsiveness (the Pitanga platform responsiveness) decays as the emulated digital circuit gets bigger and bigger. The same behavior is expected as the emulated circuit frequency increases. Figure 1.3 aims to identify and locate a set of pre-defined circuits following a meaningful methodology. In order to define what is the acceptable system responsiveness to the user, this thesis borrows the latency concept from computer games. In computer games, latency refers to the amount of time between an observable cause and its observable effect (GLAZER; MADHAV, 2015). So, for a digital circuit running in the Pitanga platform, latency is the amount of time it takes between an input change (i.e., push button, clock rising) to be observable at the output (i.e., LEDs, displays).

Finally, after identifying the emulated circuit designs that do not compromise the usability of the system due to latency, the thesis correlates the emulated designs with past commercial IC designs with similar transistor counts. The goal is to demonstrate that the Pitanga platform can emulate digital circuits complex enough to be used as a virtual remote laboratory, benefiting educational institutions with a limited budget.

1.4 Thesis Organization

This thesis is divided into six more chapters (seven in total, considering the introduction). The role of the additional chapters in the organization of this thesis is described in the following.

Chapter 2: *GENERAL CONCEPTS* — presents the theoretical foundation and established knowledge that is needed to understand the Pitanga platform. The chapter discusses VLSI design, relevant EDA data structures, simulation techniques, multiplayer game programming concepts, hamming codes and hamming distance.

Chapter 3: *REVIEW OF REMOTE LABORATORIES* — reviews previous and recent works on FPGA-based remote laboratories. It also maps and discusses several simulation-based tools used in digital circuit laboratories.

Chapter 4: *THE PROPOSED REMOTE LABORATORY* — proposes the emulation-based remote laboratory. The architecture of the system and implementation details are discussed. The chapter discusses the predictive emulator algorithm using modern EDA data structures in order to avoid system latency.

Chapter 5: *METHODOLOGY* — describes the input and output parameters, how the input parameters are controlled and how the output parameters are measured in order to have reproducible experiments.

Chapter 6: *RESULTS* — presents and discusses the results obtained when running the emulation-based system with a pre-defined set of circuits. Results in terms of the platform latency are presented. The chapter also correlates the results with past IC designs.

Chapter 7: *CONCLUSIONS AND FUTURE WORKS* — provides the conclusions. The chapter also discusses the contributions and future works.

2 GENERAL CONCEPTS

This chapter presents the foundational concepts required to comprehend the virtual remote laboratory architecture proposed in Chapter 4. In order to understand the proposed architecture, named Pitanga platform, this thesis employs knowledge from three distinct areas: VLSI design, electronic design automation (EDA), and multiplayer gaming programming.

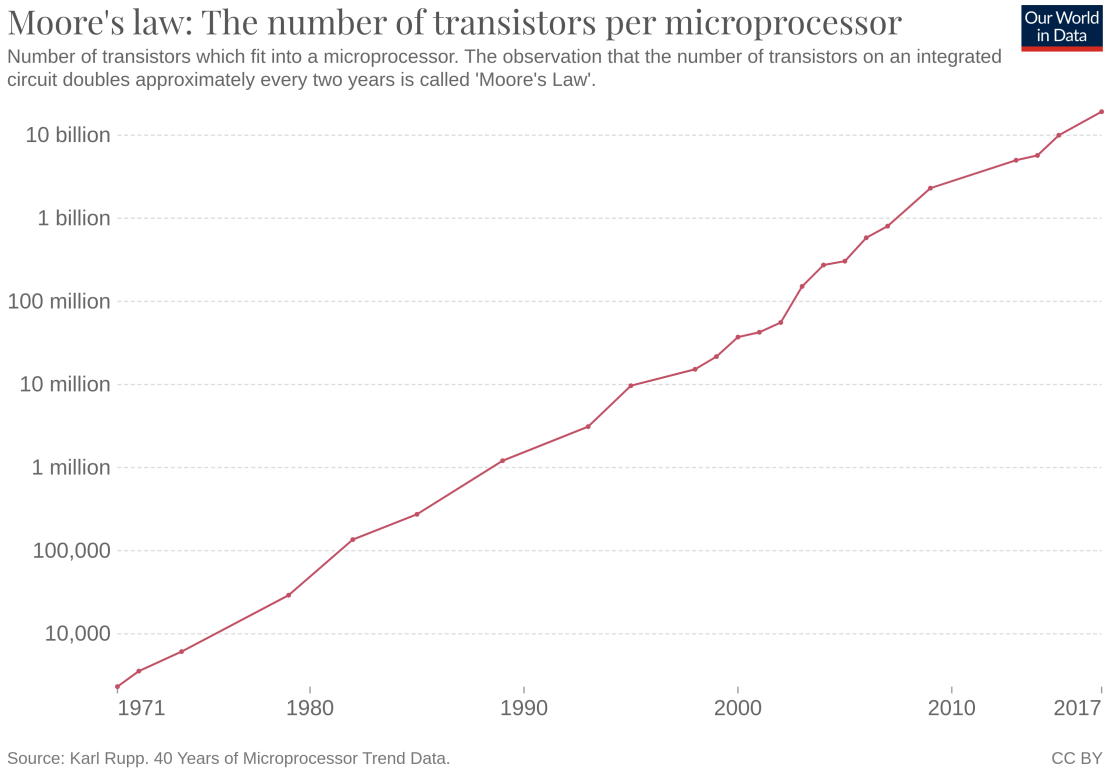
The chapter begins by explaining the key aspects of a VLSI design in the Section 2.1. This section focuses on the necessary concepts for integrating the Pitanga platform in the most appropriate VLSI design methodology and flow. Then, Section 2.2 presents modern EDA data structures in order to support the reader on how to model and emulate a virtual programmable chip in software. Finally, Section 2.3 explores multiplayer gaming programming to help the reader understand the importance of the client-server architecture and the predictive algorithm proposed in Chapter 4.

2.1 VLSI design

VLSI design is the process of building a digital integrated circuit through the connection of electrical components in a meaningful way in order to achieve an expected functionality. The expected functionality usually starts with a set of statements written in human language, which is gradually transformed into a network of transistors (RABAEY; CHANDRAKASAN; NIKLIć, 2003). VLSI stands for Very Large Scale Integration.

The increasing miniaturization of VLSI technologies raised the complexity of designing digital circuits. Since the invention of the IC in 1958 by Jack Kilby and Robert Noyce (SMIL, 2018), the transistor count in a single die has followed the prediction proposed by Gordon Moore (aka, Moore's law), doubling the number of transistors in a chip every 18 months (MOORE, 1965) as show in 2.1. However, doubling the IC integration capacity does not mean doubling the engineers' capacity for designing and verifying new digital circuits every 18 months. This productivity steep is only possible with good design methodologies and more advanced EDA tools. For this reason, successful VLSI design, like any other engineering project, demands efficient methodologies and tools.

Figure 2.1: Moore's law: The number of transistors per microprocessor. Although it is not a law, Moore's prediction is commonly referred to as a law in VLSI design. SOURCE: (OURWORLDINDATA, 2022)



There are different methodology styles depending on the chip requirements. The VLSI design methodology choice affects the VLSI design flow, which is the sequence of steps necessary for achieving a successful VLSI design. The following sections describe these design methodologies as well as the typical design flow used in modern VLSI designs.

2.1.1 VLSI design methodologies

When a company is designing a digital IC, different design methodologies can be used. These methodologies impact design factors like cost, complexity, power, schedule, performance, flexibility, and the company business (WESTE; HARRIS, 2010). The choice of a methodology determines the overall approach on how to perform the design steps that leads to a successful IC design. The comprehension of the available design methodologies also aids the company in defining the suite of EDA tools necessary for designing a chip.

The following sections briefly describe the design methods used to implement

a VLSI design. Understanding such methods will let the reader understand the methodologies which the Pitanga platform is best suited for.

2.1.1.1 FPGA-based design

Companies use an FPGA-based design methodology when the volume of chips is not high enough to justify the mass production of chips. This design method employs an off-the-shelf chip containing an array of logic cells surrounded by programmable routing resources (HUTTON; BETZ; ANDERSON, 2016). The FPGA is programmed (or configured) through a binary file known as bitstream. The design software accompanying the FPGA builds the bitstream file according to the design intent, usually described in Verilog, VHDL, or schematic.

FPGA stands for Field Programmable Gate Arrays, meaning that the chip is programmable in the field (i.e., in the customer). In order to possess such programmable characteristics, the FPGA contains logic blocks composed of registers and multiplexers (XILINX, 2016). Each register connects to the inputs of the multiplexer. During the power-on time, the FPGA bootstrapping circuitry configures the registers through the bitstream file. Thus, depending on the register content, the multiplexer assumes a different logic function. In other words, the multiplexer emulates any logic gate in a programmable manner. The number of logic gate inputs is the number of the selection inputs of the multiplexer.

The programmable feature of FPGAs has the advantage of speeding up the validation of a product in the market. Depending on the number of sold pieces, the FPGA-based design can migrate to a cell-based design in order to decrease the unit cost.

2.1.1.2 Standard cell-based design

A standard cell-based design relies on a set of pre-designed logic cells with a wide range of functionality. The logic cells implement functionalities such as boolean algebraic gates of various inputs (inverters, buffers, 2-input NANDs, 3-input XORs, and others), adders, comparators, multiplexers, laches, flip-flops, registers, and other more advanced logic circuits (SKYWATER, 2022). These logic cells are grouped into a library of cells that employs industry-standard file formats like Liberty, LEF (Library Exchange Format), and Verilog simulation files. Foundries and vendors offer

libraries (aka, technology libraries) to the market targeting a specific technology.

In a standard cell-based design, the specified digital circuit (usually written in Verilog) is mapped to the cells in the Liberty file. The produced circuit is a gate-level netlist containing a network of cells, and this netlist proceeds to physical design after verification. The physical design extracts the cell geometry and pin locations from the LEF file to drive cell placement and wire routing. The engineers check each implementation step with verification tools such as simulators and emulators in order to preserve the functionality of the design. Verilog simulation files are necessary for simulators, especially in simulations considering timing delay effects. Although not mandatory, the technology library may also include power/ground cells (PG pads), reset cells (XRES), test/analog pads, and input/output cells (IO pads) in addition to the standard cells.

Most VLSI designs are standard cell-based, as the freedom provided by full-custom designs is difficult to be efficiently used in practice (KANG; LEBLEBIGI, 2003). The cell library simplifies the design because the designer does not need to design logic gates and size every transistor. However, the designer is restricted to the use of the cells from the technology library. This approach applies primarily to ASICs and, as the final result, produces a set of fabrication masks that is sent to the foundry for manufacturing the final IC.

2.1.1.3 Gate array-based design

This method uses a set of fabrication masks containing a base array of either transistors or logic gates (WESTE; HARRIS, 2010). The specified digital circuit is designed by using the base array of pre-placed cells fixed by the already-produced masks. The functionality of the design is achieved by the routing wires connecting the gates in the base array. In gate array-based designs, the design costs reduce because only metal and via masks need to be manufactured.

From the designer's perspective, the gate array-based designs use practically the same set of EDA tools. The difference comes at the physical design level, which limits the freedom of the designer to routing the wires among the already placed cells on the floorplanning. IO pads are also fixed, thus, also limiting the input and output capabilities of the design. In addition to that, designs demanding memory blocks can only be implemented using such a method if memory blocks are available in the floorplanning.

Gate array-based methodologies have the advantage of speeding up the fabrication process because manufacturing starts from the bottom metalization layers and above. These layers apply to the premanufactured wafers, reducing the turnaround time to a week or less (RABAEY; CHANDRAKASAN; NIKLIć, 2003). Although this methodology is less expensive than the standard cell-based which does not need a complete run, this comes at the expense of lower performance, lower integration density, or high power dissipation.

2.1.1.4 SoC-based design

SoC-based design is a methodology that started appearing in technical books in the middle of the 90s (GOOGLE, 2022). The term SoC stands for System-On-Chip, and the key aspect of this methodology is the reuse of pre-designed functional blocks, named Intellectual Properties (IPs), for building the final chip (CHAKRAVARTHI, 2019). The SoC-based design is somewhat similar to building-block toy assembling (i.e., LEGO), where pre-constructed blocks are used to assemble a larger design. The IP may be company-owned or licensed by another company for reuse. The licensing company receives royalties for that.

An SoC-based design differs from other VLSI methodologies because they may integrate digital and analog IPs in the same IC design. This IP may be either soft or a hard IP. In soft IPs, the designers can access the behavioral code written in Verilog or VHDL format. This characteristic allows RTL design engineers to generate gate-level netlists similar to standard cell-based designs. However, when the design contains hard IPs, designers do not have the behavioral code and must use LEF files to place and route the hard IP with the other IPs in the digital system. Hard IPs have fixed sizes, imposing design engineers to define the hard IP locations in the floorplanning.

In the present day, most VLSI designs are SoCs of considerable complexity. Thanks to the reuse of IPs, modern IC are in the range of hundreds of billions of transistors. The Apple M1 Ultra, released in March 2022, integrates 114 billion transistors and is such an example. This SoC contains 20-core CPUs, 64-core GPUs, and 32-core neural engines (APPLE, 2022). However, in order to reach such advancement, SoC-based designs demand new EDA tools for qualifying, verifying, and integrating IPs. They also demand lots of computing power and efficient EDA tools to sign off the chip under an appropriate design schedule.

2.1.1.5 Full-custom design

In this methodology, the designer can modify the dimensions of every transistor and create any logic cell because the designer is no longer limited to the standard library cell. However, transistor sizing is time-consuming and requires experienced design engineers in transistors and circuit theory.

Full-custom design uses much fewer EDA tools when compared to gate-array and standard cell-based methodologies. As the design is manual, there is no need for too much automation. Thus, the tools used in full-custom designs are practically the same as the ones used in analog IC designs. Tools like layout editors, parasitic extractors, design rule checkers, and analog simulators are enough for this methodology.

The full-custom design is a common approach in high-volume ICs production because the sales volume amortizes the intensive engineering resources (DILLINGER, 2019). This is the case for arithmetic circuits in general purposes processors and special blocks that requires either high-speed or low-power consumption (or both). Standard cell libraries also demand characteristics such as high-speed, low-power consumption, and compacted area. For this reason, standard cells also use the full-custom design method. Nowadays, a complete chip design using this methodology from scratch is unfeasible, the reason why other VLSI methodologies are preferable.

2.1.1.6 Final remarks

The previous sections explained design methodologies where the platform proposed by this thesis can be used for teaching purposes. However, not all VLSI design methodologies are suitable for the Pitanga platform, which is the case of the full-custom design methodology.

It is important to note that VLSI designs can also mix different methodologies. For example, an SoC design that integrates a RISC processor with buses and memory blocks may need standard cell logic to control such blocks; this same SoC may also need an additional customized block designed for low power. In this example, the SoC design employs SoC-based, standard cell-based, and full-custom design methods in the same design.

2.1.2 VLSI design flow

The more significant VLSI designs become in transistor count, the more complex the design process is. The design process is composed of several steps that continuously grow with the advance of foundry technologies. The number of design steps grows due to several factors, which include:

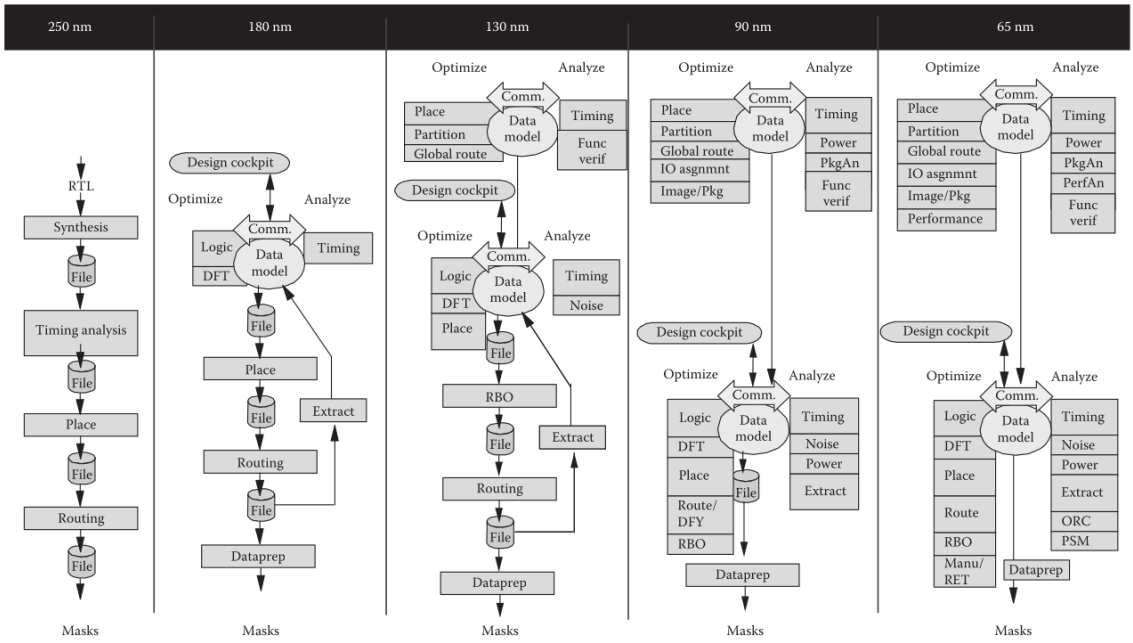
1. the increasing number of integrated circuits per millimeter square, which demands more sophisticated tools to address multi-billion transistor designs (see Figure 2.1);
2. the reuse of IPs, which demands special tools and computer power for IP qualification, integration, verification, and chip sign-off;
3. the finding of new physical effects, which demands the modeling of new parameters for the IC fabrication process;

In order to accelerate the next generation of VLSI designs, EDA vendors collaborate directly with foundries (CADENCE, 2021; SYNOPSYS, 2022c; SIEMENS, 2022a). Based on the new technology challenges, these vendors develop new EDA software and features capable of automating the new design steps. The vendors then turn the new design steps into software and release them to the design companies. The design companies, willing to advance their products to the new technology, must update their software, qualify the new design tool flow with the new technology, review the computing infrastructure, and train the engineers, among other tasks. This process repeats over and over with every new foundry technology released.

Figure 2.2 overviews the design steps necessary in different technologies ranging from 250nm to 65nm (CHINNERY et al., 2017). Notice the growth in the number of design steps as technology advances. Each new design step is associated with a new EDA tool that automates the flow. Sometimes, the new design step is so complex that it may even demand a new job position. For example, the DFT engineer, the STA engineer, and the IP integration engineer are typical job positions nowadays that did not exist in the past.

Note that a VLSI design flow details the design methodology into several steps. Therefore, VLSI design flows vary depending on the selected design method. As an example, the design flow in Figure 2.2 details a standard cell-based design methodology into several steps.

Figure 2.2: Design flow steps on different target technologies. SOURCE: (CHINNERY et al., 2017)



The following sections describe design flow steps and industry-standard data formats used in a typical VLSI design flow. It also differentiates design flows according to the design method whenever necessary. This thesis does not intend to make an exhaustive list describing every VLSI design step and data format. Instead, the following sections intend to overview common VLSI design steps to help the reader understand how the Pitanga platform can be used with a typical VLSI design flow.

2.1.2.1 Product specification document

Any VLSI design starts with market research. The marketing team compiles the research into a document named *market requirements document*, or MRD (CADENCE, 2008c), which outlines the market needs. With the MRD, the product team develops the *production specification document* (PSD) containing technical details about the product, including the foundry technology and expected die area. Product functionalities, preliminary bill of materials (BOM), and possible third-party suppliers (IPs, standard cells, EDA software licenses) may also be included in the PSD for the product cost estimation. Both MRD and PSD are written in natural language, and the names may vary from organization to organization.

2.1.2.2 Architecture specification document

The *architecture specification document* (ASD) is a technical document that contains further details about the IC architecture to be implemented. This document provides information about block partitioning, functional and performance requirements, design phases and deliverables, IPs (memory blocks, processors, standard cell library), and the design implementation flow (CADENCE, 2008a). The design manager is responsible for writing this document for the IC designers in human language. This document is very technical; it is the starting point of the IC design, and it is based on the PSD.

2.1.2.3 Verification plan

The verification plan (VPlan) is the document that details the tests that must be performed to validate the ASD. While the ASD defines the expected functionality of the IC, the verification plan defines the verification strategy to validate the IC (CADENCE, 2006). The plan addresses the list of design features and specifies the testcases that verify each feature. It also defines the verification methodology, the verification environment, and the coverage metrics used to determine the acceptance criteria of the IC design.

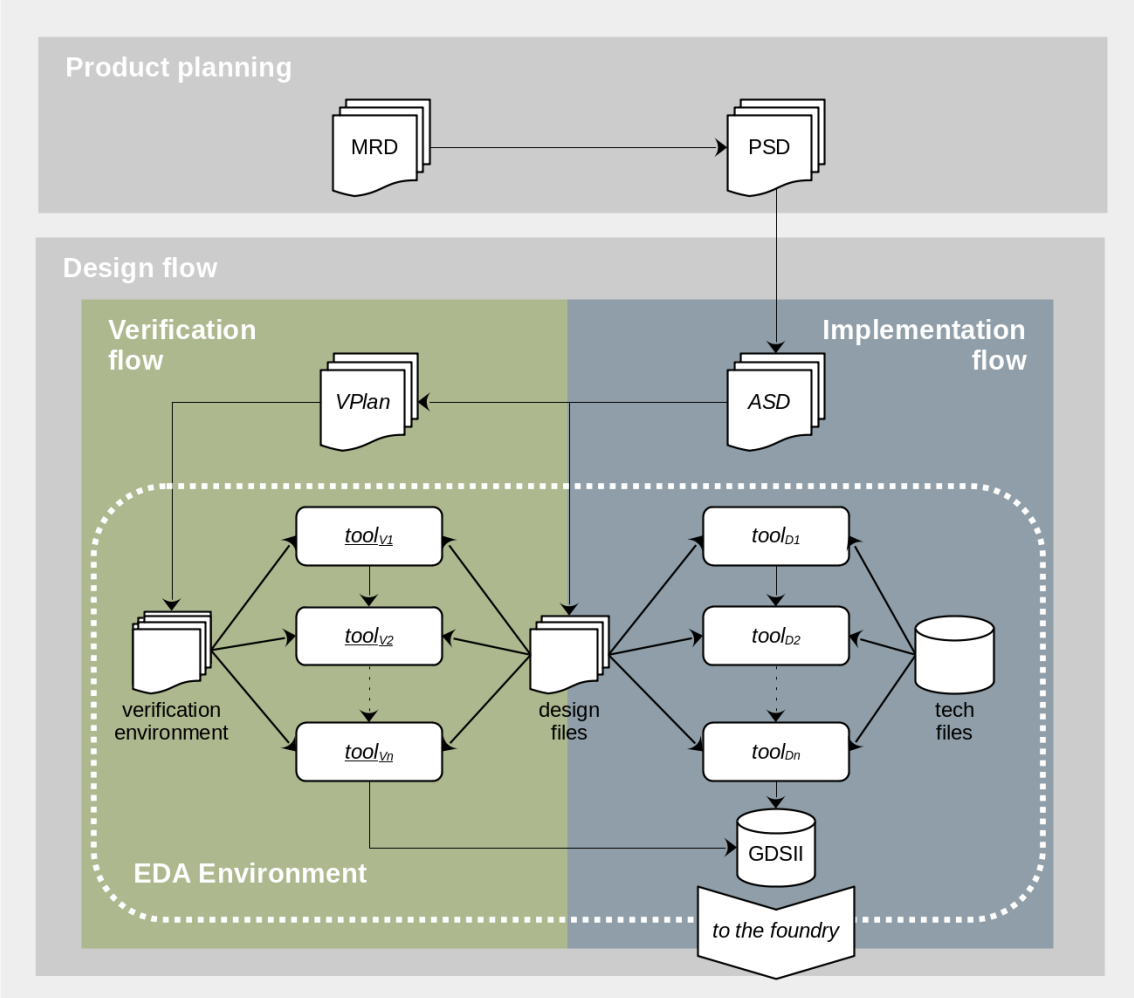
Depending on the design methodology, decisions about the verification flow are determined by this plan. For smaller designs, functional and gate-level simulations may be enough for signing off standard cell-based designs chip. However, hardware-accelerated simulations may be needed for bigger designs. This includes software licenses for parallel simulation dispatched on either in-house grid engines or third-party hardware-accelerated devices (SCHIRRMEISTER; BERSHTEYN; TURNER, 2017). Emulator and FPGA-based prototyping boards are specially employed for verifying SoC-based designs containing embedded software. Formal verification and assertion-based techniques may also be defined in this plan.

The verification plan may also be found in the technical literature as verification test plan, or just test plan. The verification manager writes this document in human language based on the ASD.

2.1.2.4 EDA Environment

The EDA environment is the implementation of the design flow. The CAD team, based on the target technology in the PSD, the design steps in the ASD, and the verification methodology in the VPlan, is in charge of building and supporting this environment. As the design flow comprises the implementation and the verification flow (CADENCE, 2008a), the CAD team is responsible for integrating and qualifying both flows with the target technology and EDA tools. Figure 2.3 shows the EDA environment and all the steps detailed so far. Notice that the EDA environment may be viewed as a container for the EDA tools, technology files, verification environment, and design files.

Figure 2.3: An VLSI design flow at the highest level including product development processes



The CAD team works in collaboration with the design team in order to qualify the tech files (foundry target technology, IPs, VIPs, standard cell, and other files) with the EDA tools. The qualification step is crucial because it speeds up time-

to-tape-out by identifying potential issues early in the design flow. For example, it is not uncommon to detect the same cell in the timing library (liberty file) and the simulation library (either Verilog or VHDL) with a cell name mismatch (CADENCE, 2007b). Or yet, pin names mismatches between back-end views from the same OpenAccess database. Sometimes, proprietary technology files are not provided by the foundry PDK, demanding the generation of such files with specific tools. This process aims to validate the consistency and completeness of the technology libraries and is called library qualification.

The qualification process is not limited only to the library qualification; it extends to the qualification of the EDA tools and the computing infrastructure as well. Some tools may require OS library updates, which may affect the operation of other tools if applied without caution. In addition to that, verification tools demand lots of computing power for simulation. If not properly qualified with the infrastructure, they may slow down all the computers in the same network.

The EDA tool qualification process serves to identify potential infrastructure bottlenecks early in the flow, such as insufficiency of CPU cores, storage, switches, or memory. There is no use in buying high-speed storage if there are not enough computing cores, as there is no use in buying more computing cores if the storage is not fast enough. There must be a balance between storage writing speed and computing power to avoid bottlenecks.

The same rationale applies to computing power and software licenses. There must be a balance. For example, having high-speed storage, network, and CPU cores makes no sense if there are insufficient EDA software licenses. All of this must be considered in order to have an efficient EDA environment.

2.1.2.5 Verification Flow

The verification flow contains scripts that launch different verification tools depending on the *design under test/design under verification* (DUT/DUV) development stage (CADENCE, 2007c). The CAD team works in collaboration with the verification engineering team in order to automate the verification flow.

As the verification tools demand lots of simulation computing power, the CAD team develops scripts to optimize the usage of the available software licenses and computing resources throughout the organization's projects. When several teams work on different IC designs, the CAD team is also responsible for defin-

ing the usage policies of hardware-accelerated simulation tools (e.g., third-party emulators and compute grid engines).

The verification flow is commonly related to the set of logical simulation tools. However, there are other types of verification tools in this flow. For example:

- Static timing analyzers (STA) verify the design timing (CADENCE, 2008e);
- physical verification tools verify the geometry of the design against the IC fabrication rules (CADENCE, 2008g);
- power verification tools verify the dynamic power consumption of the design (CADENCE, 2008f);
- constraint checkers verifies the consistency of synthesis design constraints files (CADENCE, 2007a).

This thesis considers that the verification flow consists of more than just logic simulation tools and their respective files. Although these tools and files must be considered in Figure 2.3, they are not shown in order to keep the schema clear.

2.1.2.6 Implementation Flow

The implementation flow contains the tools and technologies required for implementing the IC Design (CADENCE, 2008a). The CAD team works with the design engineering team to automate the implementation flow through shell scripts.

After library qualification, the implementation flow is validated with a sample design. This step may require additional software patches and OS library updates. The CAD team may also identify tool bugs, reporting a fix to the EDA tool vendor. The vendor solution may take some time, and, for this reason, as for the verification flow, it is recommended to validate this flow in order to avoid unexpected errors during the design implementation phase.

Much like the verification flow, the implementation flow is an ongoing process that continues throughout the IC design flow. This flow requires additional software, such as language compilers, script interpreters, FPGA design software, PCB editors, mathematical software (e.g., MATLAB), office packages, version control systems, bug trackers, and additional pieces of software necessary for the project.

2.1.2.7 Verification Environment

The verification environment is the implementation of the verification strategy defined in the verification plan. SystemVerilog is the preferred language for coding verification environments nowadays, Although SystemC, Python, C/C++, and Matlab are also used (MOURSI et al., 2018). The verification environment follows the methodology and methods defined in the verification plan.

Much like the IC designer for IPs, verification design engineers reuse past verification environments in order to accelerate future designs. For this reason, several verification methodologies attempts started after the 2000s for easiness the reuse, such as eRM (eReuse methodology), AVM (Advanced Verification Methodology), OVM (Open Verification methodology), and UVM (Universal Verification Methodology) (ROSENBERG; MEADE, 2013). The UVM has become the standard IEEE 1800.2; consequently, many companies employ UVM for verifying VLSI designs.

Nowadays, EDA vendors licenses *verification intellectual properties* (VIPs) in the same manner as IPs. While IPs are intended for companies willing to accelerate VLSI designs by reusing functional pre-designed blocks, VIPs are intended for companies willing to accelerate VLSI designs by reusing certified verification environments to validate their IPs. In summary, VIPs are a good strategy for saving the development time of a new verification environment from scratch.

2.1.2.8 Design files

Design files implement the hardware architecture specification expressed in the ASD, capturing the design engineer's intent into a file format. Although design files are commonly associated with *hardware description languages* (HDL), there are also design files describing low-power structures, maximum path delay, and maximum allowed logic area. These are the constraint files that, combined with HDLs, capture the whole design intent.

When describing the logic functionality, VHDL and Verilog are the languages that come to the design engineer's mind. VHDL was conceived at the beginning of the 80s for documenting purposes (SHAHDAD et al., 1985), while Verilog was conceived in the middle 80s as a language for circuit simulation (MAGINOT, 1992). Nowadays, both languages are used for circuit simulation and synthesis. These languages can describe digital logic in structural and behavioral forms (WESTE;

HARRIS, 2010). The structural form is very similar to a textual description of a schematic, while the behavioral format can describe circuits at higher abstraction levels (i.e., RTL and architectural levels). For this reason, the behavioral format enhances productivity and thus is the preferred method for describing circuits.

Constraint files do not implement the logic functionality but the circuit performance requirements defined in the ASD. These files drive EDA tools toward the final chip mask, constraining the circuit area, the allowed propagation delay between flip-flops, and the maximum power consumption, just to name a few examples. The *synopsys design constraint* (SDC) is the most prominent constraint file and is also the industry standard for writing timing constraints. At the time this thesis is being written, Synopsys controls the format and provides SDC as an open-source format (GANGADHARAN; CHURIWALA, 2014; SYNOPSYS, 2022b).

Table 2.1 shows industry-standard design files commonly used in the VLSI design flow (ACCELLERA, 2022). Except for SDC (SYNOPSYS, 2022b), all the other formats are IEEE standards.

Table 2.1: Design file standards used in industrial VLSI design flows

Language	**Usage**
Verilog	HDL. Describe digital circuits in RTL, gate-level, and transistor-level. The RTL and gate-level notation is commonly used by VLSI and FPGA synthesis tools.
VHDL	HDL. Like Verilog, however, it cannot describe circuits at the transistor level. Used by VLSI and FPGA synthesis tools.
PSL	Property specification language. Formal notation for specification of electronic system behavior. Capture design intent in a form suitable for verification tools.
SystemC	HDL. Describe complex heterogeneous systems that are hybrid between hardware and software. Mostly used in HLS tools.
SystemVerilog	HDL. Subsumes Verilog and enhances the language by supporting advanced verification structures. Used by verification tools.
UPF	Constraint file. Specify power intent for energy-aware VLSI designs. Enable portability of power intent across EDA tools.
SDC	Constraint file. Used mainly to describe timing and drive RTL synthesis and physical design tools throughout the VLSI design flow.

2.1.2.9 RTL simulation

In VLSI design, simulation is the process of reproducing the behavior of an electronic circuit in a computer environment. The electronic circuit is modeled with the help of HDLs, which are then read and translated by EDA simulation tools. In a simulation, the verification environment drives the DUT with computer-generated stimuli (i.e., non-real traffic). Depending on the purpose of the simulator, circuit characteristics such as functionality, timing, and power are analyzed. When the purpose of the simulation is to reproduce the logic functionality, the term functional verification is used (SCHIRRMEISTER; BERSHTEYN; TURNER, 2017).

RTL simulation is a functional verification where the DUT is described in the RTL form. The same HDL files used for the RTL simulation are used for the RTL synthesis, the reason why this design step triggers RTL synthesis as shown in Figure 2.4. RTL simulation, also known as pre-synthesis simulation, is characterized depending on the host architecture where the DUT runs as follows:

- **Traditional RTL simulation**: the main vehicle for the functional verification of digital blocks and IPs of low transistor count. It runs on general-purpose x86 CPUs and is employed early in the verification flow when hardware bugs are frequent. The term RTL simulation usually refers to this type of simulation.
- **Parallel RTL simulation**: speed up traditional RTL simulations by running several instances of the DUT in parallel. This type of simulation requires a grid scheduler software (ORACLE, 2010) for dispatching DUTs and testbenches to different x86 cores over the network (PULLI; KREMASTIOTIS; KULIS, 2022). Companies with plenty of EDA licenses and a computer farm typically use this type of simulation.
- **Hardware accelerated RTL simulation**: in this type of RTL simulation, the DUT runs on a hardware accelerator while the verification environment runs in the designer's workstation. The acceleration is achieved by mapping the DUT into the hardware accelerator, which may be composed of GPUs (ZHANG; REN; KHAILANY, 2020), FPGAs (BIANCOLIN et al., 2019; ALDEC, 2022), or customized processors (CADENCE, 2022d). Typically, acceleration can reach, or even exceed, 1000 times over traditional RTL simulation (SCHIRRMEISTER; BERSHTEYN; TURNER, 2017).

Although not mandatory, traditional and parallel RTL simulations are enough

in standard cell-based designs. Hardware-accelerated RTL simulation is preferred when the gate count impacts the simulation speed, which is usually the case of SoC-based designs.

2.1.2.10 In-circuit emulation

Emulation is the imitative process of the functioning of one hardware system by using another hardware system with the same functionality. The emulated hardware behaves identically to the original hardware system, although it contains hardware pieces that are not physically identical to the genuine hardware system. In VLSI design, emulation is also referred to as *in-circuit emulation* (ICE). Like RTL simulation, ICE is also a functional verification step because it aims to reproduce and check the logic functionality. The input design files are described at the RTL abstraction level, meaning that RTL simulators and synthesis tools can use the same files.

ICE maps the RTL design files into specialized hardware that interfaces with a prototype of the target system. The specialized hardware emulates the forthcoming IC of the target system, allowing system-level and software testing prior to the silicon availability. ICE differs from RTL simulation because the verification environment is not modeled in software, but it is in-circuit. Real electronic devices interfaces with the emulated DUT, thus, providing live real-time traffic in the verification process (MACMILLEN et al., 2000).

Figure 2.4 shows ICE as a functional verification step of SoC-based design. ICE is generally employed in SoC-based designs because the gate count is much higher when compared to standard cell-based designs. The usage of IPs is the leading cause of such gate count difference, demanding more powerful verification tools. As an SoC contains several IPs, some orchestration is needed to control the system. Therefore, SoC-based design usually includes an embedded microprocessor. The microprocessor demands software, requiring hardware and software co-development in the design flow. ICE lets software developers verify the software behavior in the target system during the design flow, thus accelerating the development cycle.

Depending on the host architecture where the DUT runs, emulation is characterized as follows:

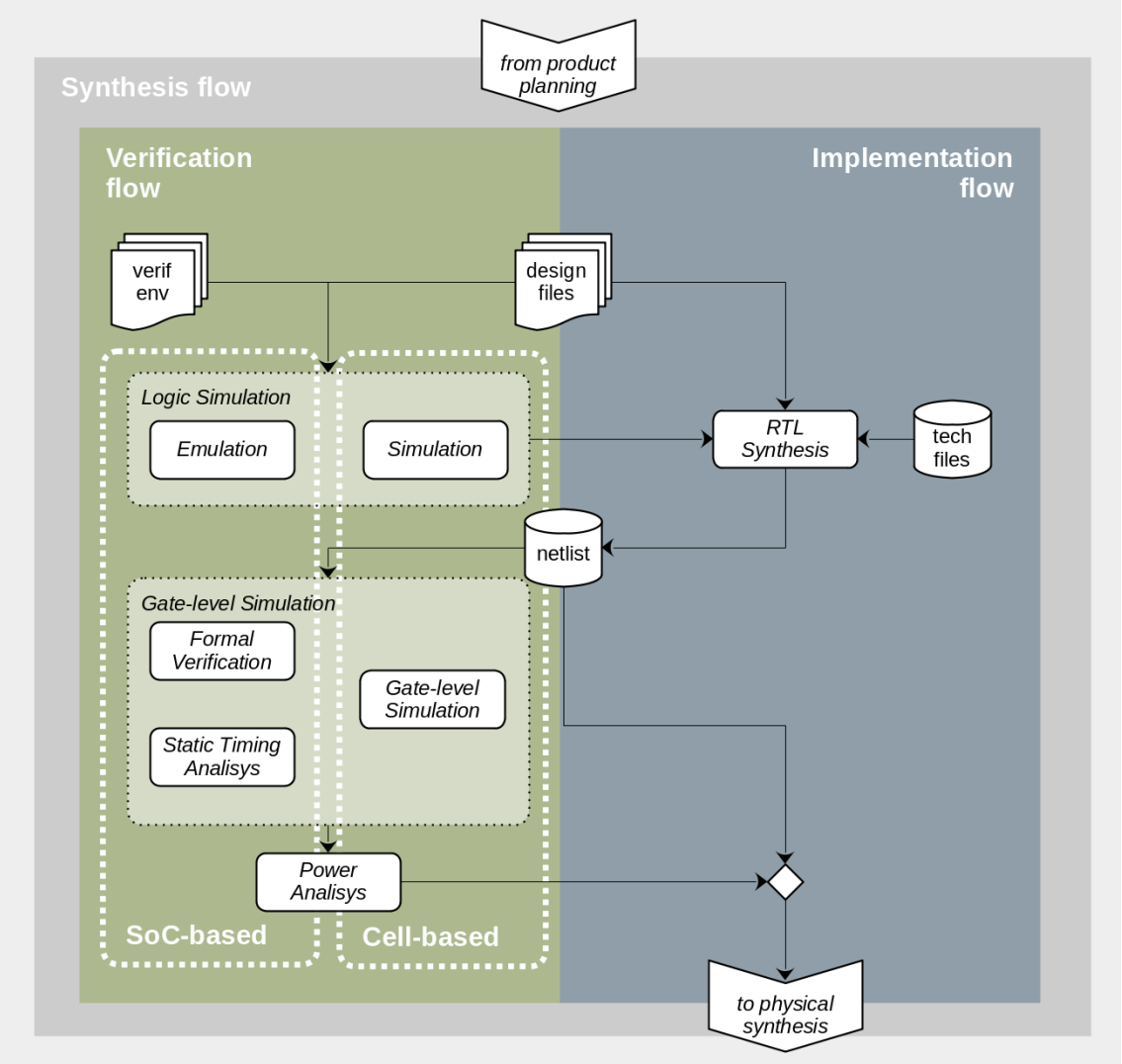
- **Processor-based emulation**: consists of a massive array of boolean pro-

cessors connected through inter-processor switches (PFISTER, 1982). Each boolean processor is a RISC architecture with logic and arithmetic instructions that simulates logic gates (BACHRACH et al., 2017). The processor-based emulator comes with a compiler that maps and partitions the design files among the several boolean processors. The compiler also schedules individual boolean operations in the correct time sequence among the boolean processors, thus emulating the intended logic circuit. Hardware debugging is straightforward, much like in RTL simulators. Due to their fast compilation time, hardware and software developers use processor-based emulators early in the design flow when hardware bugs are frequent. As of the writing of this thesis, the Cadence Palladium was an example of a processor-based emulation platform (CADENCE, 2022c).

- **FPGA-based emulation**: consists of one or more off-the-shelf FPGAs combined into a single emulation platform. The emulation platform maps and partitions (i.e., compiles) the design files among the FPGA devices. Depending on the partition cut, wires connecting two partitions may exceed the number of ports of a single FPGA, requiring repartitioning and remapping. For this reason, the time to recompile a large design is very long and discourages changes (SCHIRRMEISTER; BERSHTEYN; TURNER, 2017). Hardware debugging is an offline process and requires the definition of probe signals before compilation. Although FPGA-based emulation has some disadvantages, this platform is much faster when compared to processor-based emulators. For this reason, software developers prefer FPGA-based emulators for software debugging toward the end of the development cycle, when hardware bugs are less frequent. The Synopsys ZeBU and Siemens Veloce are examples of commercial FPGA-based emulators (SYNOPSYS, 2022a; SIEMENS, 2022b). The term *FPGA-based prototyping* is commonly used for these commercial emulators. In this thesis, the term FPGA-based emulation is preferable.

Finally, unlike RTL simulation, unexpected or unplanned stimuli are easily captured in ICE, exercising the DUT in states not even imagined by the verification team. Hence, there is a parcel of randomness in ICE that RTL simulation cannot replicate. For example, simply pressing a button by a user may produce results in an emulator that an RTL simulator cannot reproduce.

Figure 2.4: VLSI synthesis flow overview



2.1.2.11 RTL synthesis

The RTL synthesis translates a digital circuit from the behavioral level to the logic gate level (WESTE; HARRIS, 2010). The RTL synthesis tool infers the digital circuit intent coded by the design engineer and maps it to the logic gates made available by the liberty file (aka technology library). The resulting output is a list of logic gates, a network of interconnected gates, named gate-level netlist or just netlist. The netlist is usually written in Verilog format and is a textual representation of a circuit schematic.

The technology library generally contains complex logic cells in addition to primitive gates. The library has cells of different sizes, speeds, power, and functionality. This variety of cells is used by RTL synthesis tools depending on the constraint files. Constraint files, like SDC and UPF, drive the synthesis process to choose the

most appropriate cells to achieve the requirements defined in the ASD. For example, if the timing constraints are aggressive, the synthesis tool maps the design to high-speed cells, thus increasing the total area. Alternatively, if power constraints are specified, the synthesis tool may choose low-power cells from the tech library. When no constraint is explicitly defined, the RTL synthesis tool selects smaller cells because a smaller area means less die area, consequently, less cost.

Depending on the EDA vendor, RTL synthesis tools may also provide the following functionalities (CADENCE, 2008d):

- **Scan chain insertion**: replace flip-flop cells with scan flip-flops from the tech library. It also connects the scan flip-flops serially to form a scan chain. Automatic test equipment uses the scan chain to test the chip for failures after manufacturing.
- **Architecture selecion**: generates the best architecture implementation for datapaths using the cells in the tech library. For example, the tool may map to a carry-save adder instead of a ripple-carry adder.
- **Low-power synthesis**: if provided by the tech library and defined in the power constraints file (UPF), the synthesis tool inserts low-power cells such as clock gating, state retention power gating (SRPG), isolation, and level shifters in the design.

The output of the RTL synthesis is a gate-level netlist and an updated SDC file. These files are the inputs to the physical design. The RTL synthesis runs several times until functional verification does not find hardware bugs anymore.

2.1.2.12 Formal verification

Formal verification mathematically proves whether two digital circuits have the same boolean functionality (WESTE; HARRIS, 2010). This characteristic of formal verification is a valuable resource for comparing the several transformations that occur with the design files throughout the VLSI design flow, from RTL to GDSII. A formal verification tool can prove that both RTL (i.e., pre-synthesis) and gate-level netlist (i.e., post-synthesis) are equivalent without running simulation/emulation tools. It can also prove the logical equivalence of the design after applying (CADENCE, 2007b)

1. DFT transformations, such as scan chain insertions.

2. logic optimizations, such as resource sharing.
3. low-power techniques, such as clock gate insertions.
4. ECOs, such as fixing a hardware bug after silicon prototyping.

When combined with STA tools, formal verification reduces the need for time-consuming gate-level simulations (GLS). As formal verification techniques rely on mathematical operations, input stimuli are unnecessary. For this reason, formal verification is more efficient than GLS when comparing two different versions of the same design. However, GLS is still necessary for signing off the chip.

2.1.2.13 Static timing analysis

Static timing analysis (STA) computes and compares every path delay in the gate-level netlist against the timing specification defined in the SDC file (CADENCE, 2008e). The path delay, also named *timing path*, is the propagation time of data from a start point to an end point relative to the circuit clock. A start point is either an input port or a clock pin of a sequential cell, while an end point is either an output port or data input of a sequential cell. Unlike gate-level simulation, STA does not need input stimuli to verify the timing correctness of the design. Every timing path is computed with the pre-computed cell delays available in the Liberty file. When combined with formal verification, STA reduces the need for time-consuming gate-level simulations.

As the project moves toward the end, accurate propagation delay in the wires becomes available. This accuracy occurs after physical design, which places and routes the circuit components (i.e., standard cells) and extracts the wiring delays. Physical design tools annotate wire delays into a file named *standard delay format* (SDF) (DILLINGER, 2019). With the SDF available, the designer can run a much more precise STA, which is closer to the real and expected timing. The analysis also considers the signal propagation at different temperatures and voltage operating conditions. These conditions also apply to temperature and voltage variations in different parts of the chip (on-chip variation, OCV). This type of analysis aims to identify the best and worst-case operating conditions of the designed circuit and is also known as corner analysis.

2.1.2.14 Gate-level simulation

Gate-level simulation (GLS) reproduces the behavior of the DUT after the RTL synthesis. In GLS, the same verification environment applied in functional verification stimulates the DUT. However, in this case, the verification environment uses only some testcases because gate-level simulators are extremely slow for computing gate delays in large circuits. Thus, running all the testcases would be very time-consuming (SINGH, 2015).

Although the couple STA and functional verification (FV) are more efficient than GLS, this couple cannot certify the design's correctness in some specific scenarios. One example scenario occurs when RTL synthesis inserts low-power structures to shut off/power-on entire circuit portions. These structures cannot be easily verified with the STA/FV duo, especially during the transition from and to these states. Another common scenario where GLS is well-suited is during the analysis of power-up and reset states. These states take several clock cycles; non-expected glitches may occur and impact the circuit behavior (EDN, 2014).

Notice that VLSI design uses the same term *gate-level* simulation for post-synthesis and post-layout simulations. Post-layout GLS differs from post-synthesis GLS because it computes both gate and interconnection delays in simulation. This simulation is very precise, is used for signing off the chip, is compute-intensive (i.e., needs computer farms), and is also known as regression.

2.1.2.15 Power Analysis

Power analysis estimates the power consumption of the circuit, which is composed of both dynamic and static power (WESTE; HARRIS, 2010). Dynamic power is proportional to the signal-switching activity, while static power primarily depends on the current leakage of CMOS transistors. Design engineers reduce the dynamic power consumption with low-power circuit structures, while static power is reduced with specially designed low-voltage threshold CMOS transistors. A modern SoC design contains multiple cells of different threshold voltages to reduce static power (CHAKRAVARTHI, 2019).

Power analysis tools use the signal-switching activity of VLSI designs to estimate power. RTL simulator generates this signal-switching activity from a particular set of test vectors and stores it into either the industry-standard file VCD or

SAIF (IEEE, 2019; IEEE, 2018). The switching-activity file, in combination with the UPF file that guides the circuit implementation, analyzes whether or not the specified power consumption is met. If power consumption is not met, the circuit is redesigned. Design engineers may also decide to pack the die into a different package material, i.e., ceramic or plastic, or use an external cooling system for the chip (CADENCE, 2008a).

2.1.2.16 Physical Design

The goal of the physical design is to produce the physical layout of the target circuit. The main input files that guide the physical design are the post-synthesis netlist, the power and timing constraints (UPF and SDC), and the timing and physical libraries (Liberty and LEF). The GDSII file is the output of the physical design process.

Physical design is out of the scope of this thesis, and for this reason, this large process is summarized as follows (WESTE; HARRIS, 2010; CADENCE, 2008b):

- Floorplanning: define the die size, IO and core boundaries, the location of blocks for placement, and blockage areas. Add power rings and power stripes to connect blocks and cells to the power structures.
- Placement: place standard cells in the floorplanned design. It also specifies spare, JTAG, and padding cells for placement. Clock buffers replace padding cells during clock tree synthesis.
- Scan Reorder: reconnects the scan chain implemented during RTL synthesis, considering the scan flip-flops location after placement. Reorders the scan chain for optimizing the routing resources and timing.
- Clock tree synthesis: distributes the clock signal, inserting clock buffers to minimize the arrival latency and signal skew among the various sequential cells distributed over the chip.
- Route: route the interconnections among standard cells, I/O cells, and IPs to match the post-synthesis netlist. Use the corresponding metal layers in order to meet the timing constraints defined in the SDC file;
- Extraction: calculate the parasitic resistance and capacitance from the interconnections. Write it into a file named SPEF. Timing and power analysis tools use SPEF files to generate more accurate analyses.

- Delay Calculation: compute the delay of every interconnection and standard cell in the design. Write the information into the SDF file, which is used by STA tools.
- Signal Integrity: analyze unintended noise effects in the signal propagation caused by either parasitic resistance or capacitance, such as signal glitches due to long nets routed in parallel.
- IR Drop: check whether the chip power supply does not drop below acceptable voltage levels because of abrupt power consumption changes, causing unwanted chip malfunction.
- EM Analysis: check whether the current density in all parts of the chip does not exceed specified levels, resulting in either open-circuits or short-circuits.
- GDSII Export: transform the placed and routed design to a file containing the geometries of every electrical component. The foundry uses this file to build the mask set that will manufacture the chip.
- Layout-Versus-Schematic: compares the transistor-level netlist with the GDSII file to ensure the connectivity of the design. This step is also known as LVS.
- Design Rule Check: compares whether the physical layout geometries in the GDSII file follow the technology drawing rules provided by the foundry. This step is also known as DRC.

2.1.2.17 EDA Files

EDA tools output different file formats throughout the VLSI design flow. Each format has a purpose; some of them are proprietary, while others are industry standards. These files are frequently modified by EDA tools throughout the flow. Table 2.2 shows common EDA files and their corresponding usage in modern VLSI design flows (CHAKRAVARTHI, 2019; CADENCE, 2008b):

Table 2.2: Files generated by commercial EDA tools throughout the VLSI flow

Format	Usage
DEF	*Design Exchange Format*. Industry-standard. Generated by place and route tool. Contain floorplanning information such as die size, IO and core boundaries, and logical connectivity.
SDF	*Standard Delay Format*. IEEE stardard 1497-2001. Contain delay information of cells and nets. Produced by timing calculators after place and route. Used by STA and simulators.
SAIF	*Switching Activity Interchange Format*. Defined in the IEEE standard 1801-2018. Contain the switching activity of signals captured by simulation tools. Used by power analysis tools.
HDL	Verilog and VHDL files. As defined in Table 2.1. EDA implementation tools output modified gate-level netlists of these files throughout the flow.
VCD	*Value Change Dump*. Defined in the IEEE standard 1800-2017. Contain information about value changes that occur during the simulation of selected signals of the design.
GDSII	*Graphical Design System II*. Industry-standard. Contain the design layout in the form of planar geometric shapes. Used by DRC and LVS tools before signing off the design for mask production.
SPEF	*Standard Parasitic Exchange Format*. IEEE standard 1481-2019. Communicate parasitic information throughout the design flow process. Used by circuit simulators and power analyzers.

2.1.2.18 Technology Library

The technology library contains the cells necessary for the standard cell-based methodology. This library is composed mainly of analog, logic, and I/O cells. Foundries or IP providers design these cells, usually using the full-custom based methodology. The tools in the implementation flow use the cells from the technology library to map the design described from the behavioral level to the gate-level, and from the gate-level to the physical level.

The files in the technology library are also generated by EDA tools. However, these files are not modified during the implementation flow. For the scope of this

thesis, the following technology files are of interest:

Table 2.3: Technology files in a technology library

Format	Usage
HDL	Verilog and VHDL files. As defined in Table 2.1. Contain information about the logic functionality of every cell in the library. Simulators use these files in order to run gate-level simulations and regressions.
Liberty	*Liberty*. Industry-standard. Property of Synopsys Inc. Contain power, timing, and area information of every cell available in the technology library. Used by RTL synthesis, power and timing analyzers.
LEF	*Library Exchange Format*. Industry-standard. Property of Cadence. Contain physical information about the geometries of every cell available in the technology library. Used by physical design tools during placement.

2.2 Electronic Design Automation

Electronic Design Automation (EDA) is the discipline that builds software that automates the circuit design steps. It allows designers to simulate and test the functionality of the design before physical implementation, which helps identify potential issues early on. This section presents the data structure and briefly discuss different digital logic simulation techniques that support the work of this thesis.

2.2.1 And-Inverter-Graphs

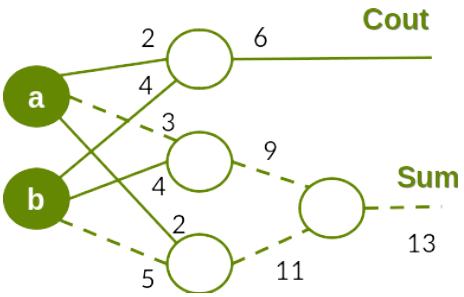
And-Inverter Graphs (AIGs) are directed acyclic graphs where the internal nodes represent 2-input AND gates. The edges between the nodes represent wires and may also contain inverter gates, enabling AIGs to represent any Boolean function using AND and NOT logic operators (WANG; CHANG; CHENG, 2009).

The efficiency of AIGs originates from the uniformity of their nodes, which ensures that all nodes have the same number of inputs and outputs. This uniformity facilitates merging nodes sharing the same logical conditions, resulting in a compact

and memory-efficient data structure. Additionally, representing AIGs as vectors of integers enables iterative algorithms to be used instead of recursive algorithms, further improving their efficiency.

Figure 2.4(a) shows a half adder represented as an AIG. Next to it, in Figure 2.4(b), the same AIG is represented as a vectors of integers. The nodes represent AND operations and the dashed edges represent NOT operations. Note that each edge of the graph has two associated numbers. These numbers represent the input signals of a node and are described in Table 2.4(b) by the columns $i0$ and $i1$, respectively. The index column represents the node where the signals $i0$ and $i1$ meet in Table 2.4(b). The value of the index is calculated by performing integer division on the number associated with the edge leaving the same node that the input edges meet. Therefore, as an example, node nine is represented by the index $9div2 = 4$, which has as inputs the numbers $i0 = 3$ and $i1 = 4$. Node six is represented by the index $6div2 = 3$, which has as inputs the numbers $i0 = 2$ and $i1 = 4$. Node four is represented by the index $5div2 = 2$, which has as inputs the numbers $i0 = 4$ and $i1 = 4$. As the inputs are equal, this means a primary input.

Figure 2.5: Example of a half-adder represented as an AIG



(a) Example AIG

Index	i0	i1
1	2	2
2	4	4
3	2	4
4	3	4
5	2	5
6	9	11

(b) AIG as a list of vectors

The exception to this rule is the primary inputs of the AIG, represented by repeated even numbers in the $i0$ and $i1$ columns.

These characteristics make AIGs an unique and efficient data structure for Boolean function representation and reasoning, making it the data structure of choice for modern logic synthesis (REIS; DRECHSLER, 2018) when compared to truth tables, sum-of-products, product-of-sums, and binary decision diagrams(BDDs).

2.2.2 Zero-delay simulation

Zero-delay simulation is a type of digital circuit simulation technique that assumes the delay of each gate to be zero, meaning that the output of the gate changes immediately after the input changes (GEREZ, 1999). This assumption simplifies the simulation process, making it faster and more efficient. It is particularly useful in analyzing digital circuits that operate at high clock frequencies or have many stages, such as microprocessors and memory circuits.

However, it is important to note that the zero-delay simulation does not consider the physical delays associated with the propagation of signals through the circuit (CADENCE, 2008a). As a result, it may not accurately reflect the actual behavior of the circuit, especially for designs that require precise timing analysis or that operate at low frequencies.

2.2.3 Unit-delay simulation

Unit-delay simulators take into account the delay of each gate in the circuit. The output of a gate changes after a certain delay, which is determined by the gate's propagation delay (GEREZ, 1999). The simulator models the circuit as a sequence of discrete time steps, with the output of each gate updated at the end of each time step. This approach accurately represents the circuit's behavior but requires more memory and computational resources (CADENCE, 2008a).

Unit-delay simulation is computationally more expensive than zero-delay simulation because it requires the evaluation of the propagation delay in every input or internal state change. Therefore, it is typically used for detailed analysis of digital circuits or for verifying circuits that require accurate timing behavior.

2.2.4 Event-based simulation

Event-based simulators are designed to simulate the DUT only when input stimuli from the verification environment changes (WANG; CHANG; CHENG, 2009). The simulator processes these stimuli, known as events, at the specific time of each event using a queue structure.

Following the queue schedule, the simulator computes and propagates the events to the DUT outputs. This propagation updates every circuit gate and sequential cell, including combinational and sequential circuits. The system clock is also considered an event in this type of simulator.

Despite being slower than cycle-based simulators, event-based simulation is widely used in the EDA industry (CADENCE, 2008a). It provides a significant advantage in supporting digital circuits in different levels of abstraction, such as behavioral, RTL, and gate-level models. The time between events and the circuit size impacts the simulation time. Consequently, more events (e.g., higher clock frequency) require more computing processing.

2.2.5 Cycle-based simulation

Cycle-based simulators evaluate the inputs of the DUT only when the system clock changes (WANG; CHANG; CHENG, 2009). The inputs to the circuit are applied at the beginning of each clock cycle, and the circuit outputs are evaluated at the end of each cycle. The simulator then advances to the next cycle, repeating the process with new input values. This process continues until the desired number of cycles has been simulated or until the circuit reaches a stable state.

Unlike event-based simulators, this technique accelerates the simulation process by avoiding evaluations of intermediate events that do not impact in the circuit state. The simulator stores only the output signals of flip-flops, discarding the combinational logic output. This approach results in a reduced memory footprint and faster simulation time, as it is proportional to the number of cycles and flip-flops (CADENCE, 2008a). However, its usage is restricted to synchronous circuits only.

2.3 Multiplayer Game Programming

Multiplayer game programming is the process of creating video games that allow multiple players to interact and play together in real-time. It involves designing and implementing game mechanics, network architecture, and communication protocols to enable seamless gameplay experiences for players across various platforms. The following sections present some concepts from multiplayer game programming

that support the work of this thesis.

2.3.1 Client-Server model

The client-server model is a network topology frequently used in online games. It involves a central computer, referred to as the server, that communicates with all other computers, referred to as clients. In this model, the server must possess more bandwidth and processing power than the clients since it is responsible for communicating with all of them. The server acts as the central hub in this topology.

The client-server model is prevalent in various game genres, including first-person shooter games, action games, multiplayer massive online games, and real-time strategy games (GLAZER; MADHAV, 2015). The complexity of this topology surpasses that of online single-player games because the actions on the client-side involve the transmission of data packages to the server-side. The server is responsible for processing these incoming packages and broadcasting the changes to all clients connected to it.

In VLSI design, the client-server model is frequently employed to execute computationally demanding circuit simulations, thereby relieving the client CPU. In this situation, the server possesses substantially greater computing power than the clients, and broadcasting to all connected clients is unnecessary due to the distinct nature of VLSI design.

2.3.2 Latency

In the context of computer games, latency refers to the time delay between a cause and its observable effect (GLAZER; MADHAV, 2015). This delay can be observed in various scenarios, such as the time between a mouse click and a unit responding to orders in a real-time strategy game or between a user moving their head and the virtual reality display updating in response.

Different types of games have varying tolerance levels for latency. VR games are the most sensitive to latency, with a latency of less than 20 ms required for the user to remain immersed in the simulated reality. Fighting games, first-person shooters, and other action games are also sensitive to latency, ranging from 16

to 150 ms before the user perceives the game as unresponsive. Real-time strategy games have the highest tolerance for latency, with some games reaching up to 500ms without being detrimental to the user experience.

2.3.3 Round Trip Time

Round trip time (RTT) refers to the time it takes for a network packet to travel from the player's device to the game server and back to the player's device again. RTT is typically measured in milliseconds (ms), and it represents the delay or latency experienced by the player in sending a command to the game server and receiving the response.

A low RTT is desirable in online gaming because it means that the player's actions can be communicated quickly and accurately to the server, and the player can receive a response without experiencing noticeable delays. On the other hand, a high RTT can result in lag or other performance issues, which can negatively impact the player's experience.

Game developers and network engineers work to minimize RTT by optimizing server locations, reducing network congestion, and using other techniques to improve network performance (MADHAV, 2014).

2.3.4 Jitter

The RTT between two hosts is not a fixed value. It fluctuates over time, resulting in a deviation from the expected value. This deviation is known as jitter, which can negatively impact the overall performance of client-server applications (GLAZER; MADHAV, 2015).

Jitter affects the arrival time of network packets, changing the arriving order. A reliable transport protocol like TCP or a customized packet ordering system is necessary to guarantee an ordered packet delivery. Given the adverse effects of jitter, reducing it as much as possible is essential to improve the application's usability. Sending minimal packets to maintain low traffic and deploying servers close to clients are common techniques for reducing jitter.

2.3.5 Client Prediction

Client prediction refers to the ability of a client to forecast the future states calculated by the server (MADHAV, 2014). For instance, in an online 3D map game with multiple players, the client prediction algorithm can estimate the intermediate frames between server updates, inferring other players' positions based on their last known location and velocity.

If the server updates occur sufficiently frequently, the client's representation of other players will be reasonably accurate. However, a poor connection makes client predictions inaccurate when updates are infrequent. Since the server has the final authority, the client must correct any differences between the prediction and the actual positions. Nonetheless, if the prediction is reasonably accurate, it will appear smooth and uninterrupted to the player.

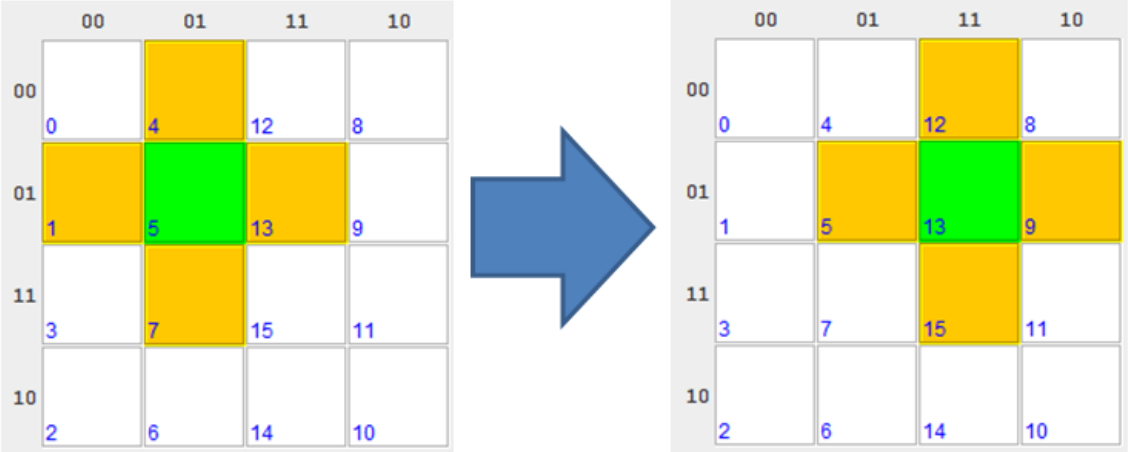
2.4 Hamming Codes and Hamming Distance

Hamming codes are a concept that will be used in the predictive emulator proposed in this thesis. This section describes the concept. Consider Figure 2.6, it shows two Karnaugh maps side by side. On the Karnaugh map on the left, position 5 is highlighted. The code corresponding to position 5 is 0101. In the Karnaugh map it is easy to see that the neighbor codes 1,4,7,13 differ only by one bit. So, it is possible to say that codes 1,4,7,13 have hamming distance 1 with respect to code 5. If only one input bit is allowed to change at a time, an eventual current input 5 can only move to next input 1,4,7,13. When the current input changes from 5 to 13 a new set of possibilities with hamming distance 1 becomes available, namely 5,9,12,15. This will be discussed later but from a basic concept perspective, the inputs in the proposed emulator can change one at a time.

2.5 Previous works from LogiCS lab

LogiCS Lab, short for Logic Circuit Synthesis Lab, is the research group located at the Universidade Federal do Rio Grande do Sul (UFRGS) (logiCS, 2023). Situated in the Informatics Institute, the lab has contributed significantly to var-

Figure 2.6: Example of Hamming code



ious research efforts. This section outlines the work of past researchers, including students and professors, who invested a portion of their lives during their academic journey at the LogiCS lab. The idea of presenting past research from LogiCS is due to the fact that the history of research in this group is at the basis of the proposed thesis.

This thesis has its roots in the foundation concepts of logic circuit design (ROSA et al., 2003; WAGNER; REIS; RIBAS, 2006) and logic synthesis (REIS; MATOS, 2018) (REIS; DRECHSLER, 2018) investigated by the LogiCS group. The continuous research throughout the years has explored various areas, including the automatic generation of logic cells for cell libraries (TOGNI et al., 2002; MARTINS et al., 2015), technology mapping (REIS et al., 1997; REIS, 1999; CORREIA; REIS, 2004), generation of circuits from BDDs (PERALTA et al., 2021; BRANDãO et al., 2022; PERALTA et al., 2023), and transistor netlist synthesis for logic cells (JUNIOR et al., 2006; da Silva; REIS; RIBAS, 2009; ROSA et al., 2007; BUTZEN et al., 2010a; BUTZEN et al., 2012; ROSA et al., 2009).

The research group also proposed efficient techniques for synthesizing transistor networks to optimize area (POLI et al., 2003), variability (da Silva; REIS; RIBAS, 2009; BUTZEN et al., 2010a; BUTZEN et al., 2012), and power (BUTZEN et al., 2010b). Some methods proposed in the LogiCS lab, such as KL-cut-based synthesis (MACHADO et al., 2012) and functional composition-based synthesis (MARTINS; RIBAS; REIS, 2012), are used in different applications. These include the use used in new technologies (NEUTZLING et al., 2013; MARRANGHELLO et al., 2015; NEUTZLING et al., 2015; NEUTZLING et al., 2018; NEUTZLING et al., 2019), robust circuits (GOMES et al., 2014; GOMES et al., 2015), and asynchronous

circuits (MOREIRA et al., 2014).

Although this thesis does not contribute with new logic synthesis methods, it absorbs the expertise and the scientific approach preserved and improved over the existence of the LogiCS lab. The works on this thesis builds on top of the prior knowledge of data structures such as And-Inverter-Graphs (AIGs). This prior knowledge built throughout the years was combined with a business need driven by inPlace Design Automation (inPlace Design Automation, 2023). This thesis innovates then as a cooperated effort, between the academia and the industry, in order to to deliver not only an academic contribution, but the proof of concept for a product that is able to emulate digital hardware in the form of software as a service (COSTA; DROVES; REIS, 2023c) to provide a virtual board approach (COSTA; SILVEIRA; REIS, 2022; COSTA; SILVEIRA; REIS, 2023) to prototype digital circuits.

2.6 Contributions of this chapter

This chapter provides the fundamental concepts necessary to understand this thesis. As the thesis presents a virtual FPGA-based board for educational purposes, it is crucial to comprehend VLSI design in order to identify that digital circuit experiments using physical FPGA-based boards are related to the in-circuit emulation (ICE) verification step. Additionally, this chapter introduced AIGs in the context of EDA because they are a very efficient data structure used to model the behavior of digital circuits in modern design tools. Finally, a few multiplayer gaming programming concepts are shown because the virtual remote laboratory proposed by this thesis employs a software architecture commonly used in online multiplayer gaming.

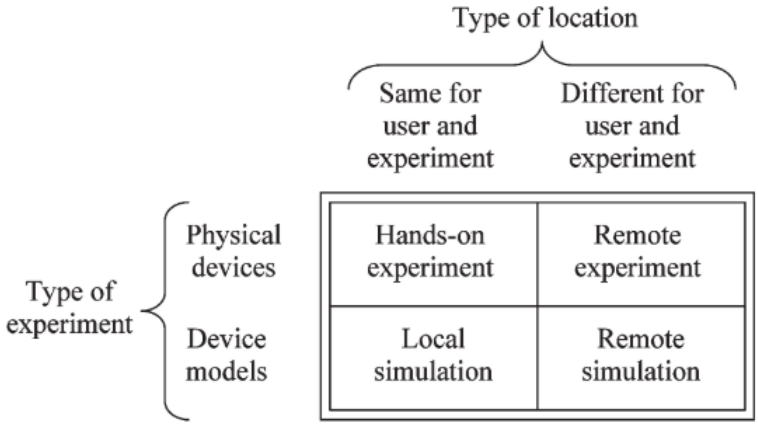
3 REVIEW OF REMOTE LABORATORIES

Engineering students need to practice the teachings they obtain during the undergraduate course. In order to provide such an experience to the students, academic professors prepare laboratory lessons to go with the theory during engineering courses. These lessons require a particular environment with physical equipment prepared and, sometimes, pre-configured to be used by the students.

The traditional way of putting into practice the concepts taught to the students is through a hands-on session, where both students and laboratory equipment are in the same physical environment. This teaching method is a common practice; however, with the advance of the Internet, remote laboratory experiments started to be used more frequently by education institutions (NEDIC; MACHOTKA; NAFALSKI, 2003). This type of laboratory differs from the traditional hands-on laboratory because the student and the laboratory equipment are geographically separated. The student accesses the lab equipment over the Internet, being able to control the intended experiment as if he or she were in the laboratory.

Considering such differences, Gomes proposed a classification scheme for laboratory experiments, represented in the four quadrants indicated in Figure 3.1 (GOMES; BOGOSYAN, 2009).

Figure 3.1: Types of engineering laboratories. SOURCE: (GOMES; BOGOSYAN, 2009)



This thesis focuses on situations where the user and the laboratory equipment (either physical or modeled) are in separate locations. Section 3.1 briefly review digital circuit experiments using FPGA-based boards remotely, while Section 3.2 presents remote simulation-based approaches. Then, Section 3.3 briefly compares both alternatives, concludes the chapter and suggests the thesis approach, which is

further explored in Chapter 4.

3.1 FPGA-based remote laboratories

FPGA-based remote laboratories are hands-on sessions where students remotely experiment with digital circuits in FPGA-based electronic boards. These FPGA devices are usually embedded in development kits containing push buttons, displays, LEDs, Wifi, USB, and other interfaces. These boards replace the old-style hands-on experiments, where the students learn digital circuit design by connecting discrete components (i.e., resistors, transistors, LEDs, TTLs) and wires into a breadboard.

The FPGA-based remote laboratory is an alternative to the traditional (i.e., in-person) FPGA-based laboratory when both students and lab equipment cannot simultaneously be physically present in the same place. A remote computer with the appropriate software configures the digital circuit designed by the student into the FPGA, which is made available over the Internet and can be accessed by the student from the student's location.

The first approach to remote laboratories was proposed by Aktan and Bohus in 1996 (AKTAN et al., 1996; BOHUS et al., 1996) in the control engineering laboratory of the Oregon State University (USA). In their works, Aktan and Bohus implemented a system for controlling a robot arm over the Internet. The students could then control the robot arm's trajectory and watch the resulting outcome by video.

Indeed, Aktan and Bohus's work was not in the VLSI design field. However, their primary contribution echoed in several engineering areas, including VLSI design. Years later, in 2003, Fjeldly and Shur (FJELDLY; SHUR, 2003) published *LAB ON THE WEB*, a book containing contributions from several researchers about running real electronics experiments via the Internet. This book is an important milestone in the history of remote laboratories in semiconductor electronics because it presented several cases from renowned universities, such as the Microelectronics WebLab from the Massachusetts Institute of Technology (MIT).

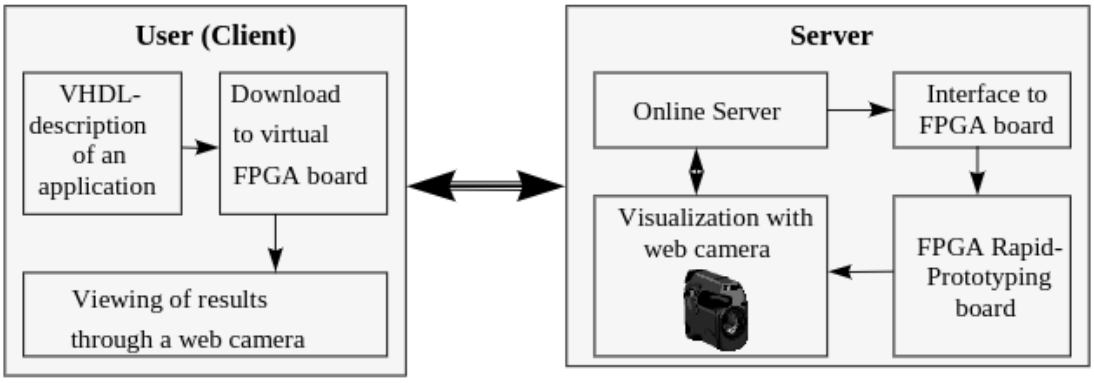
The following section summarizes the main contributions of researchers of FPGA-based remote labs. The sections were grouped according to the major contributions, as perceived by the author of this thesis.

3.1.1 Putting the puzzle together

The first laboratory experiments of digital circuits using FPGA-based boards over the Internet started very straightforwardly. During the 2000s, most FPGA-based remote labs were just remote computers sharing the screen over the Internet. Microsoft Windows was the base development platform, and both the FPGA design software and board were compatible with this platform. The difference between local and remote laboratories consisted of the visual feedback and the input interfaces. The same computer was employed for both local and remote experiments. Thus, the student used the same GUI for local and remote laboratory experiments.

The remote lab used a client-server architecture, where the server-side contained at least the FPGA configuring software and the FPGA board. Although not mandatory, the client-side might include the FPGA tool chain to implement the digital circuit and build the FPGA configuration file. Both Darmstadt University of Technology (BECKER et al., 1998; BECKER et al., 2000) and McGill University (MCCRACKEN; ZILIC; CHAN, 2003) proposed this type of laboratory around the 2000s. In their proposal, they suggested a video camera for streaming the FPGA board, providing visual feedback for the student. Figure 3.2 shows this concept.

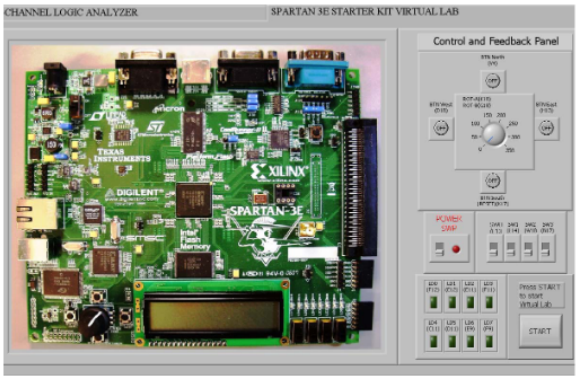
Figure 3.2: Basic FPGA remote lab architecture. SOURCE: (BECKER et al., 1998)



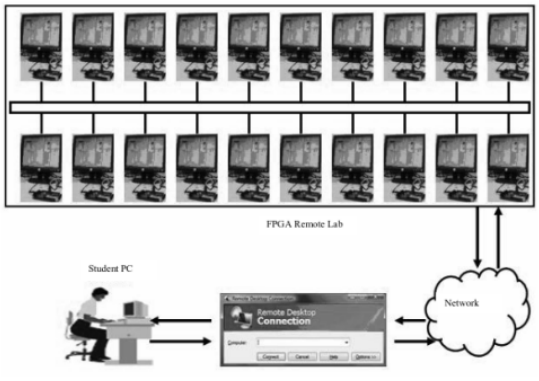
Hashemian added the missing video camera to his FPGA-based remote lab implementation (HASHEMIAN; RIDDLEY, 2007). In addition to that, in Hashemian's proposal, the FPGA board push buttons and switches could be controlled remotely. This innovation was possible because the remote lab employed custom control hardware to interface with the FPGA board. LabView, installed on the server-side, managed the custom control hardware and the webcam. Microsoft XP Remote Desktop (MRD) was required to access the lab environment.

El-Medany did not implement the video camera but presented an attempt to scale the solution proposed by Becker and McCracken. In his proposal, he deployed a remote laboratory consisting of 20 PCs, each of them connected to one FPGA board (EL-MEDANY, 2008). El-Medany did not implement any resource allocation control scheme, and MRD technology was also employed. Thus, the students might know the remote computer IP address before connecting to one of the remote stations. Figure 3.3 shows both Hashemian's and El-Medany's proposals.

Figure 3.3: Two different implementations of FPGA-based remote laboratories.



(a) Video feedback and input interfaces controlled remotely with LabView. SOURCE: (HASHEMIAN; RIDDLEY, 2007).



(b) Attempt to scale remote laboratories with several PCs sharing FPGA boards. SOURCE: (EL-MEDANY, 2008).

Perceiving the limitations of the proposed remote lab solutions to handle multiple users, Indrusiak et al. (INDRUSIAK; GLESNER; REIS, 2007) presented a comparative analysis of remote FPGA access strategies, evaluating more sophisticated communication technologies other than the MRD. In his study, he evaluated different network technologies for handling multiple users. He also considered the dynamic allocation of FPGA-based boards from a limited pool set. From a perspective, Indrusiak suggested a path for more advanced and automated ways to scale and manage the resources of FPGA remote laboratories.

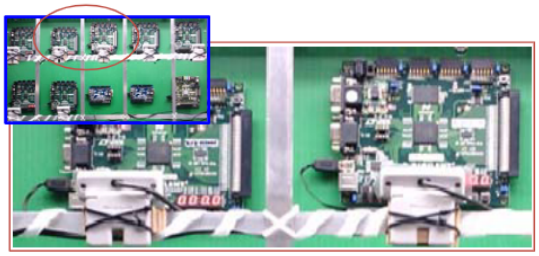
In summary, during the 2000s, the FPGA-based remote labs were nothing more than a junction of different off-the-shelf pieces of software. The challenge was "putting the puzzle together," and Microsoft XP Remote Desktop was the primary vehicle for transmitting the remote computer screen. The student should previously know the computer IP address containing the FPGA board of interest, and no resource allocation scheme was available. The 2000s witnessed the beginning of the FPGA remote labs research field, an important research topic for the succeeding years.

3.1.2 Setting up the infrastructure

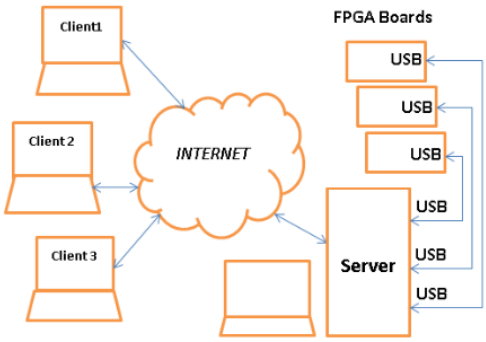
The research contributions during the 2000s showed that FPGA remote labs could not be delivered by simply joining separate pieces of off-the-shelf software. Software development was necessary to allocate the available FPGA boards to the students.

Morgan (MORGAN; CAWLEY, 2011; MORGAN et al., 2011) contributed to solving this issue by implementing an FPGA-based remote lab containing seven FPGA boards and seven webcams running concurrently. Unlike El-Madany's proposal, each user had an account that, once logged in, could access the FPGA board from a list of available devices. The system communicated with the FPGA board via a web application and USB server, using a GUI to interact with the remote board push buttons, switches, and LEDs. Ravanasa (RAVANASA; HASHEMIAN, 2014) proposed a similar approach; however, using an improved GUI and no camera. Rodriguez-Gil (RODRIGUEZ-GIL et al., 2014) improved the GUI even more, using graphical web technologies to overlay the FPGA board switches with interactive virtual switches. Figure 3.4 gives an idea of both Morgan's and Ravanasa's proposals.

Figure 3.4: Example of FPGA-based remote laboratories supporting multiple users.



(a) Multiple FPGA boards running concurrent digital circuit experiments. SOURCE: (MORGAN; CAWLEY, 2011).



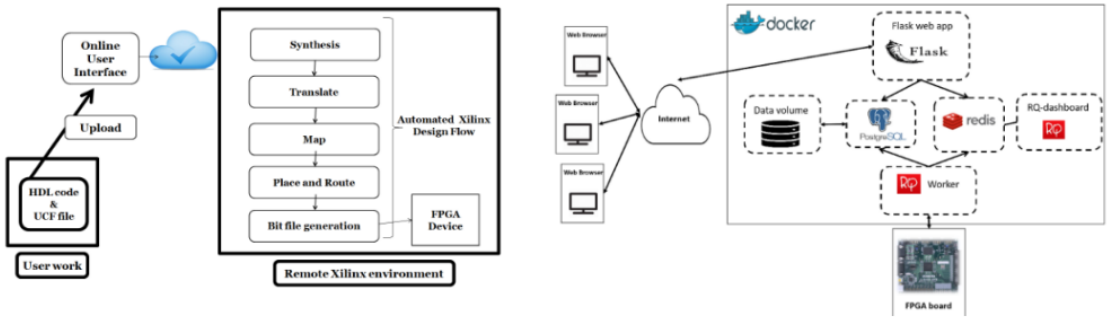
(b) Proposed architecture for supporting multiple students running experiments. SOURCE: (RAVANASA; HASHEMIAN, 2014).

The advances in remote labs have raised the question of the best location to install the FPGA toolchain because such a choice impacts the overall student laboratory experience and learning. For example, professional FPGA design software like Quartus Prime (INTEL, 2022) and Xilinx Vivado (AMD Xilinx, 2022) varies from 7GB to 70GB, depending on the software version and supported FPGA devices. The software download, installation, and licensing may seem trivial to an engineer,

but students and faculty members may have technical, computing, and personal resource limitations. In addition, students may waste considerable time downloading and installing any of these solutions on their own machines.

In order to avoid costly installations and waste of resources on the client-side, Doshi (DOSHI et al., 2015) implemented a cloud-based environment where students could run the FPGA design software on the server-side. The cloud infrastructure on the server-side relied on a LAMP stack (Linux, Apache, MySQL, and PHP), the same stack used by software as a service (SaaS) applications. Al Qassem (Al Qassem et al., 2020) employed a lightweight solution using docker containers, deploying the FPGA design flow as a cloud microservice on the server-side. Both Doshi and Al Qassem's solutions have simplified the FPGA design software access to a web browser on the client-side.

Figure 3.5: FPGA-based remote laboratories schemes using LAMP stack and microservices.



(a) System architecture using LAMP stack. SOURCE: (DOSHI et al., 2015).

(b) System architecture using microservices. SOURCE: (Al Qassem et al., 2020).

After the contributions of many researchers, Angulo (ANGULO et al., 2019) from the University of Deusto, Spain, presented a complete FPGA remote laboratory solution. The solution employed the Remote Laboratory Management System (RLMS) named WebLab Deusto, and supported the following features: user account management, user authentication, FPGA board resource selection, web cameras for real-time feedback, FIFO queue for FPGA board scheduling, web browser access from the client-side, and a virtual interface with switches and push-buttons. This solution contemplated significant contributions from past publications, setting a milestone in the developing of FPGA remote labs.

3.1.3 Scaling up the labs

The advances made in FPGA remote labs during the period described in 3.1.2 resulted in the centralization of the design software and the FPGA board on the server-side. The centralization motivated researchers to investigate new challenges to scale up FPGA remote labs. In 2018, Angulo et al. examined previous FPGA remote lab implementations and identified key technical requirements to pursue in order to deploy and maintain scalable remote labs (ANGULO; RODRIGUEZ-GIL; GARCIA-ZUBIA, 2018). These technical requirements are:

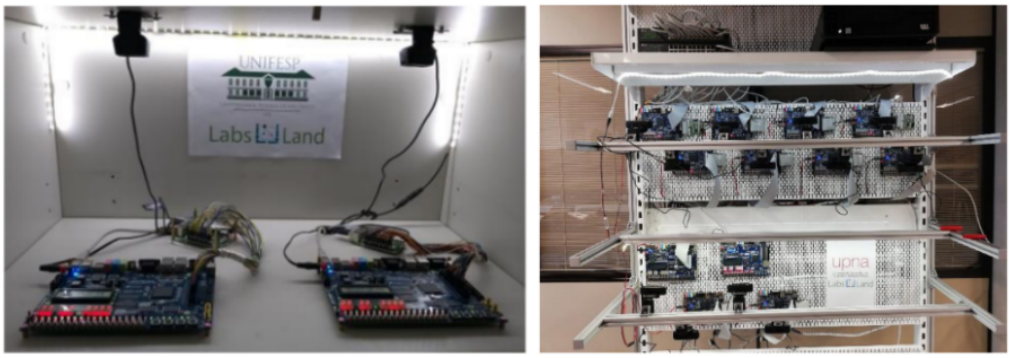
- Scalability - the remote laboratory must allocate and release the FPGA-based boards dynamically (e.g., design software, prototyping board, additional hardware) in order to support multiple users running the remote experiments;
- Adaptability - the remote laboratory must upgrade new instances without redesigning the remote lab from scratch (the cost of adding a new instance or adapting an existing one must be taken into account);
- Deployability: access to the laboratory should comply with the security policies of the institutional IT services without requiring specific configuration (e.g., open ports, firewalls, deployments of specific software);
- Universality: the student may access the remote laboratory from any device (e.g., laptop, tablet, smartphone) with any operating system in any web browser;
- Integrability: the capability of the remote laboratory system to integrate with educational platforms such as Google Classroom and Moodle.

In 2019, the University of Deusto, in conjunction with LabsLand[1], paved the road for scaling remote labs up by proposing a cost-effective architecture for embedded systems (VILLAR-MARTINEZ et al., 2019). This effort resulted in a novel laboratory architecture supporting the integration of laboratories from different universities around the globe. Recently, the UNIFESP and UPNA (Universidad Pública de Navarra) joined the Labsland laboratory network (MAYOZ et al., 2020), integrating their FPGA laboratories into one lab. The joint collaboration resulted

[1]The efforts to scale up remote labs resulted in two spin-off companies: the LabsLand from the University of Deusto and the ViciLogic from the University of Ireland (LABSLAND, 2022; VICILOGIC, 2022). The LabsLand hosts remote laboratories for several experiments, such as biology, chemistry, electronics, engineering, physics, robotics, and technology. ViciLogic offers FPGA-based remote laboratories only.

in a cross-national remote laboratory where the students of both institutions can select any of the 17 FPGA boards from both laboratories transparently. Figure 3.6 shows the laboratory infrastructure of both institutions.

Figure 3.6: The cross-national FPGA remote laboratory between Brazil and Spain. The image on the left shows the Brazilian laboratory at UNIFESP. SOURCE: (MAYOZ et al., 2020)



Despite the technological advances in FPGA remote labs, these solutions still suffer from a lack of reliability (VILLAR-MARTINEZ et al., 2021). Without fault-tolerant measures or a permanent lab support staff, the laboratory may become unavailable frequently. Consequently, the student may become confused about the reliability of his/her experiments over the Internet, particularly when the physical FPGA board is running remotely. For this reason, the latest research on FPGA remote labs focuses on finding counter-measures alternatives to improve the lack of reliability naturally imposed by hardware (VILLAR-MARTINEZ et al., 2022). One of the proposed solutions goes toward a dataset of previously recorded experiments (e.g., a set of videos and photographs of the experiment), which are streamed to the student during the lab experiment in an interactive manner (NEUSTOCK et al., 2021; ZAMAN; NEUSTOCK; HESSELINK, 2021). This approach is attractive because it improves the experiment's reliability and scalability; however, it comes at the cost of less flexibility to the student.

3.2 Simulation-based remote laboratories

Simulation-based remote laboratories provide the computer infrastructure for simulating logic circuit experiments remotely. This type of remote lab contains the simulation software installed on the server-side; the client-side usually contains a lightweight installed software or a web browser to control and observe the running

simulation on the server-side (BALAMURALITHARA; WOODS, 2009).

The most straightforward way of implementing simulation-based remote laboratories is by sharing the remote computer's screen with the software of interest installed. For this reason, little research is found on this subject, particularly for client-server logic circuit simulators. In addition, whenever possible, students may prefer to have the simulation software locally installed on his/her machine, making it unnecessary to begin any research in this field. Thus, it is more common to find research on novel EDA algorithms rather than new software architectures or better user interfaces.

However, professional-grade digital circuit simulators became highly complex, demanding complicated and long software setups to simulate simple digital circuits such as adders and multiplexers. The software setup may take considerable time in laboratory sessions, shifting the student's attention to understanding the simulator operation instead of the digital circuit experiment. Educational companies are turning this difficulty of using professional-grade circuit simulators into an opportunity to develop easy-to-use digital circuit simulators for aspiring engineering students. This trend is also motivated by the lack of engineers to fulfill technology jobs (PADWICK et al., 2020), the reason why it is becoming increasingly crucial to develop educational software to attract young students to the VLSI design career.

Table 3.1 shows several digital circuit simulators available for teaching digital circuits. In order to make a broad comparison among the simulators, the table classifies the simulators according to the following characteristics as described in their official websites:

- Arch: whether the software architecture is standalone or SaaS;
- Entry: determine whether the software inputs are non-standard schematic files or industry-standard HDLs;
- License: whether the software is proprietary or opensource;
- Purpose: whether the software is for professional or educational usage.

Table 3.1: Simulation-based software used in VLSI education. The table does not show SPICE simulators as these are out of the scope of this thesis (e.g., MultisimLive, CircuitLAB). The reference for each software is available in Appendix.

Software	Arch	Input	License	Purpose
CircuitVerse	SaaS	schematic[2]	MIT	educational
logic.ly	standalone	schematic	proprietary	educational
LogiSim	standalone	schematic	GPL2	educational
Deeds	standalone	schematic[3]	freeware	educational
Icarus	standalone	vlog/vhdl	GPL2	professional
Verilator	standalone	verilog	LGPL3	professional
LogicCircuit	standalone	schematic	freeware	educational
simulator IO	SaaS	schematic	proprietary	educational
Logic Gate	standalone	schematic	GPL3	educational
OpenCircuit	SaaS	schematic	GPL3	educational
SmartSim	standalone	schematic	GPL3	educational
BOOLR	SaaS	schematic	GPL3	educational
LogiJS	SaaS	schematic	GPL3	educational
EasySim	standalone	schematic	proprietary	educational
wiRedPanda	standalone	schematic	GPL3	educational
Digital	standalone	schematic	GPL3	educational
Hradla	SaaS	schematic	GPL3	educational
MAX+Plus II [4]	standalone	schematic vlog/vhdl	proprietary	professional
Intel Quartus	standalone	schematic vlog/vhdl	proprietary	professional
Xilinx Vivado	standalone	vlog/vhdl	proprietary	professional
Logic Circuit Pro	standalone[5]	schematic	proprietary	educational
EDA playground	SaaS	vlog/vhdl	proprietary	professional
#Data	SaaS	schematic	proprietary	educational

Notice that most of the simulators listed in Table 3.1 use non-standard design entries, meaning that the experiments designed in these simulators cannot be

[2]Limited support to Verilog.
[3]Export to VHDL.
[4]No vendor support (end-of-life).
[5]Smartphone app only

used in real FPGA-based laboratories. The exceptions are Icarus, Verilator, Intel Quartus, Xilinx Vivado, and EDA playground. However, as mentioned before, these simulators became highly complex for new-entry engineering students because they are intended for skilled engineers. The remaining options are MAX+Plus II, CircuitVerse, and Deeds. However, each has its drawbacks: MAX+Plus II has no vendor support and is standalone, CircuitVerse has minimal support to Verilog and cannot export designs, and Deeds is standalone, requiring a PC with Windows installed.

Although institutions commonly use simulators as a cost-efficient alternative for FPGA-based remote laboratories, simulators do not replace the need for experiments (FROYD; WANKAT; SMITH, 2012). In addition, simulators primary purpose is debugging, not prototyping. In a simulator, students observe a pulse switching from LOW to HIGH in a waveform diagram instead of a LED light-up. The metric for a successful experiment is abstract, although simulations have proved helpful in engineering education.

3.3 Contributions of this chapter

This chapter presented FPGA-based remote laboratories and software-based remote laboratories for digital circuit experiments. A historical bibliographical review of FPGA-based remote laboratories was presented, concluding that state-of-the-art FPGA remote labs still pursue to be fully fault-tolerant. In addition, these labs are only partially scalable, the reason why the most recent labs employ a pre-dataset of previously recorded experiments to provide a certain degree of scalability to the user.

Software-based alternatives were also presented as opposed to FPGA remote labs. Educational institutions use simulators as a cost-efficient alternative when FPGA labs are not viable. Table 3.1 presented several simulators, most from non-academic sources, and concluded that educational simulators do not follow industry-standards. As simulations do not replace FPGA labs, there is room for a solution combining the FPGA lab experience and the simulator's cost-efficiency. This solution must also consider the use of industry-standards and is further explored in Chapter 4.

4 THE PROPOSED REMOTE LABORATORY

The main technologies used for practical digital circuit laboratories in engineering education are: FPGA prototyping boards and logic circuit simulators. However, these technologies have limitations depending on how they are applied, as discussed in Chapter 3.

This chapter delves deeper into the limitations of both FPGA-based and simulation-based remote laboratories, which are explained in sections 4.1 and 4.2, respectively. Section 4.3 presents a new approach for remote laboratory experimentation using emulation, which combines the best features of both FPGA-based and simulation-based remote laboratories. The proposed system's architecture is described in Section 4.5. Lastly, Section 4.6 summarizes the chapter's contribution, which is also the thesis's contribution.

4.1 FPGA-based remote laboratory limitations

A recent study conducted by Vilar-Martinez of the University of Deusto (VILLAR-MARTINEZ et al., 2021) examined the quality of service provided by five remote labs that offer various experiments, including those that do not employ FPGA-based boards. The study aimed to identify the reasons for unavailability of these remote labs. The labs analyzed in the study were iSES (Czech Republic), RemLabNet (Switzerland), RexLab (UFSC/Brazil), WebLab-Deusto (Spain), and GOLDi (Ukraine).

To conduct the analysis, the laboratory experiments were performed on four different dates: 16-Apr-20, 26-Apr-20, 04-May-20, and 12-May-20. A total of 42 different experiments were evaluated on each day as follows:

- 17 experiments in iSES
- 06 experiments in RemLabNet
- 12 experiments in RexLab
- 03 experiments in WebLab-Deusto
- 04 experiments in GOLDi

Vilar-Martinez gathered the operation outcomes and categorized them based on the error type into separate Tables, 4.1 and 4.2 respectively:

Table 4.1: Percentage of laboratories according to their operativity over a number of 4 days. SOURCE: (VILLAR-MARTINEZ et al., 2021)

Operability per day	Percentage
Fully operative for 4 days	26.2%
Inoperative for 1 day	19.0%
Inoperative for 2 days	21.4%
Inoperative for 3 days	7.1%
Inoperative for 4 days	21.4%

Table 4.2: Relationship of type of errors and how they affected the analyzed remote experiments. SOURCE: (VILLAR-MARTINEZ et al., 2021)

Type of error	Percentage	Description
Experiment is offline	7.1%	The URL to access the remote lab is unavailable.
Experiment is not accessible	30.9%	The URL is available, however, the service for remote experiment in unavailable.
Experiment is not visible	35.7%	The webcam is not working properly and the student has no visual feedback of the remote experiment.
Experiment is not controllable	11.9%	The student can access and visualize the experiment; however, the experiment controls are not working properly.
Experiment is not observable	7.9%	The student can run the remote experiment; however, the experiment is not displayed correctly, or the data published in the interface is inconsistent with the results shown by the webcam.

Upon analyzing Tables 4.1 and 4.2, it is evident that the sample of remote

labs investigated in this study is not completely reliable. Although not apparent in tables 4.1 and 4.2, this study demonstrated the FPGA remote laboratory from the University of Deusto exhibited uncontrollable behavior in two out of four attempts due to failed bitstream uploads (i.e., the FPGA device could not be programmed remotely). Additionally, six out of sixteen attempts in the embedded system remote laboratory provided by Goldi, employing both FPGA and MCUs, resulted in experiment unavailability, indicating a 37.5% failure rate attributed to webcam issues (i.e., the experiment could not be visible).

These findings highlight the need for improving reliability in remote labs, which is essential for student confidence and overall satisfaction. To address this issue, Vilar-Martinez et al., in the same study, proposed a scheme to detect laboratory instance fails. So, to increase the student's confidence in using remote labs, it is necessary to detect when a particular instance fails and remove it from the pool of working lab instances. The study carried out an experiment to self-detect remote faulty lab instances and achieved the following automation:

1. early detection of malfunctioning lab instances and consequent support staff alert;
2. removal of the malfunctioning lab instances for maintenance, preventing students from experiencing failed sessions;
3. redirection of students to other working lab instances containing the same configuration of the malfunctioning instance;

Noticed that the proposed solution does not prevent, fix or reduce the number of internal failures. However, it contributes to removing failing lab instances, a handy feature when looking for scalable solutions.

With this in mind, Villar-Martinez et al. combined the contribution of previous work on lab scalability based on replicability (VILLAR-MARTINEZ et al., 2019) with fault-detection in a most recent study aiming to evaluate the effectiveness of remote laboratories using such techniques (VILLAR-MARTINEZ et al., 2022). The new study considered replicas of the same laboratory configuration to improve scalability. The replicas were part of a federation of remote laboratories in different parts of the world. The evaluation, therefore, included real data provided by the startup LabsLand (LABSLAND, 2022), responsible for managing and supporting the federation of laboratories.

The study evaluated 72,377 laboratory sessions across various institutions worldwide for a period of 736 days, with a focus on the LabsLand DE1-SoC FPGA laboratory, which is equipped with the Altera System-on-Chip Cyclone V FPGA and based on the Intel DE1-SoC development board. There were 62 remote laboratory instances, with 26 in Spain and 36 in the United States, all having the same configuration. The study suggested that it is possible to develop production-level laboratories for real-world multi-institutional usage by implementing the proposed fault-detection scheme with replicas of the same laboratory equipment. However, the proposed solution heavily relies on exact replicas of the laboratory instance, thus needing hardware and software uniformity beyond requiring a qualified team to fix eventual laboratory faults.

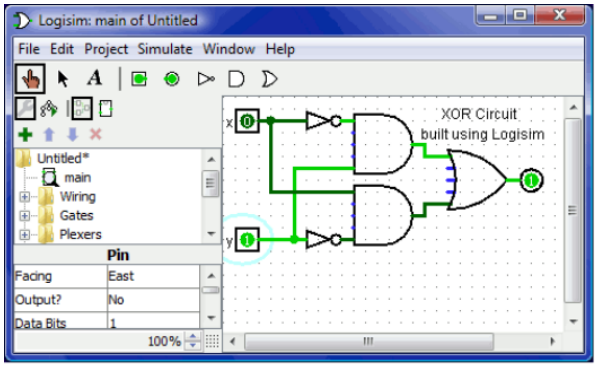
Despite the significant advances obtained by Villar-Martinez et al., the cost of FPGA-based board replication, lab deployment, and maintenance is still significantly high (VILLAR-MARTINEZ et al., 2022). In addition, the study fails to address remote laboratories' latency issues, such as the response time of students' interactions with the remote lab (e.g., a LED light-up just after pressing a button in the web browser). There is a time for the remote laboratory system to capture user interactions action in the web browser, transform it into a real signal for the FPGA-based board, and then stream it back to the user through the webcam (GUO; HUSSEIN; ORDUNA, 2022). **Therefore, in addition to the limitations imposed by the equipment costs and configuration uniformity policies, the most recent studies on FPGA remote labs did not explore the latency issues between the user and the remote laboratory**.

4.2 Simulation-based remote laboratory limitations

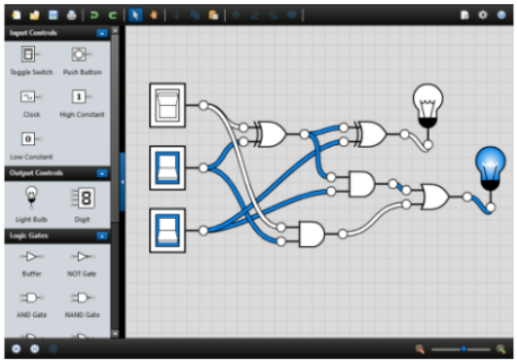
There is no physical prototyping board in simulation-based remote laboratories. The computer displays the DUT behavior on the computer screen through waveform diagrams. The parcel of randomness expected in a real environment, such as the exact moment that the user presses a button, is not considered in a simulation-based environment. As the DUT interfaces with the software-generated stimuli previously coded in the verification environment by the user, there is no real-time interaction between the user and the DUT. At first impression, the lack of real-time response sounds like a limitation.

However, some simulators provide real-time interaction. Logic.ly, CircuitVerse, and Logisim are simulation-based remote laboratories that allow students to build logic circuits with interactive virtual switches and output lights. Figure 4.1 show a screenshot as an example of such tools. Although Logisim is a standalone tool, it can be installed on a remote computer and made available over the Internet, just like the standalone tools listed in Table 3.1.

Figure 4.1: Example of simulation-based laboratories capable of interacting in real-time through virtual buttons and switches.



(a) A XOR gate design in LogiSim
SOURCE: (Carl Burch, 2022)



(b) A 1-bit full adder design in Logic.ly
SOURCE: (logic.ly, 2022)

Although some simulation-based educational tools offer real-time interaction, more is needed to guarantee a comprehensive practice experience. A hands-on experience requires the implementation of the designed circuit using the same techniques used in real FPGA-based designs. Code the design in an HDL (e.g., Verilog and VHDL), choose the target FPGA, synthesize the circuit, write constraint files, and analyze circuit logic usage, power, and timing reports to cite a few techniques. The simulation-based tools do not offer such features because they focus on simulation, not implementation.

Simulation-based and FPGA-based remote labs have different purposes. While simulation-based labs do not interact with real traffic, FPGA-based labs interact with real traffic. It is a mistake to claim that simulation-based laboratories are limited because their purpose is static debugging. And static debugging requires replicable input stimuli in order to be effective. **Therefore, the limitation of simulation-based remote laboratories occurs when educational institutions rely solely on simulators for experimenting with digital circuits.** These institutions use simulation-based tools due to the cost of FPGA-based laboratories.

4.3 Emulation-based remote laboratory limitations

Emulators are a widely explored solution in the IC design industry, but they are yet to be explored as a solution for remote laboratories. As described in Section 2.1.2.10, designers use emulators in design with many logic gates, which is why Figure 2.4 places it as part of the SoC-based flow.

Some EDA companies offer emulators as part of their product portfolio. Cadence, Siemens, Synopsys, and Aldec are among them. This small group of suppliers is due to the small demanding market niche, typically linked to few companies with sufficient capital to invest in IC projects with high sales volume in cutting-edge technologies (CADENCE, 2022a; CADENCE, 2022b). **Therefore, today's remote laboratories based on emulators are an unviable solution for education due to their high cost.**

4.4 Differences between remote laboratory types

After analyzing the limitations of FPGA-based and simulator-based remote labs in education and considering the limitations of a possible emulation-based remote laboratory, we summarize that:

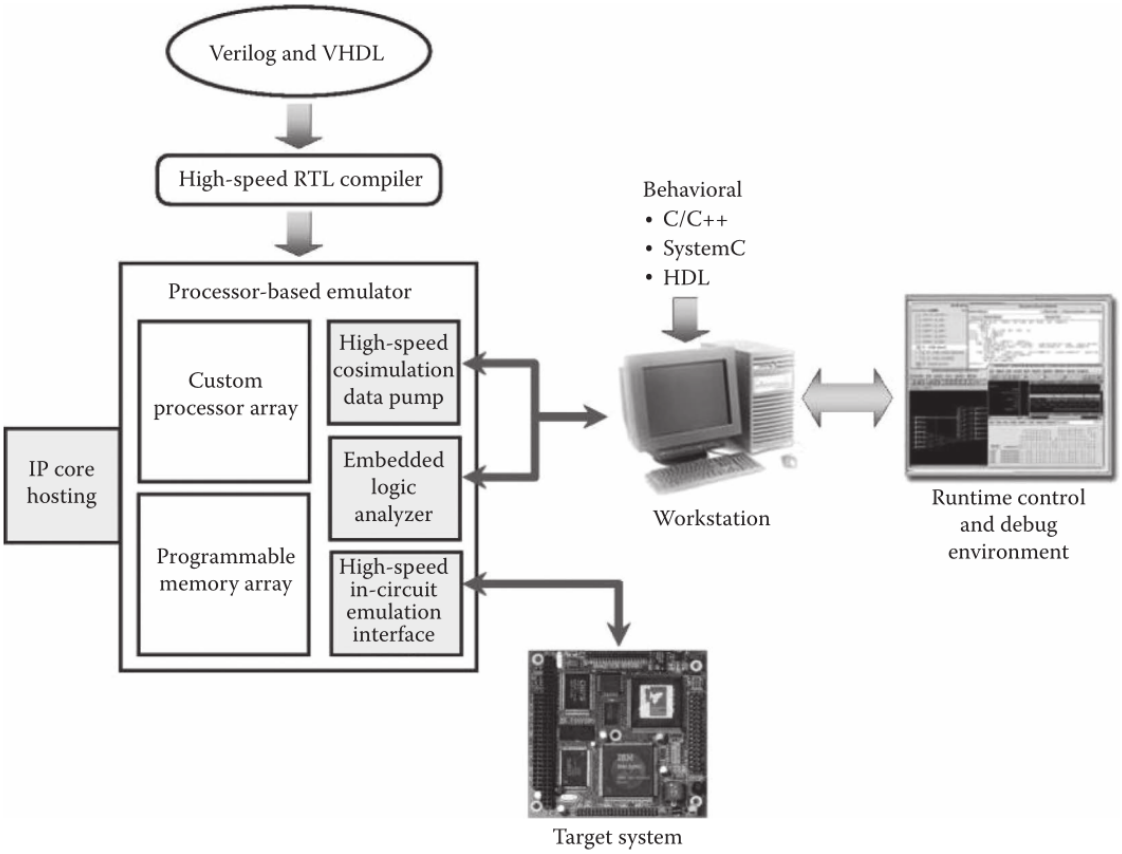
1. FPGA-based remote laboratories are expensive to implement, maintain, and scale. Also, they are not suitable for experiments that require responsiveness between the remote student and the remote board;
2. Simulator-based remote laboratories do not provide hands-on experience and, therefore, should not be used as the only alternative for digital circuit experimentation in education;
3. Emulator-based remote laboratories are not used in education, only in industry, and therefore are yet to be explored as an educational technology.

The following sections further analyze emulators to understand how this technology can help develop an emulation-based remote lab.

4.4.1 Comparison between emulation and prototying

Figure 4.2 presents a simplified architecture of an emulator being used to verify a target system. Unlike a prototype where the FPGA is directly soldered onto the board, the emulator connects to the system through an interface. The separation through an interface is one characteristic that differentiates emulators from FPGA-based prototypes. So, emulators can verify different systems, while FPGA-based prototypes are restricted to the systems they are embedded in.

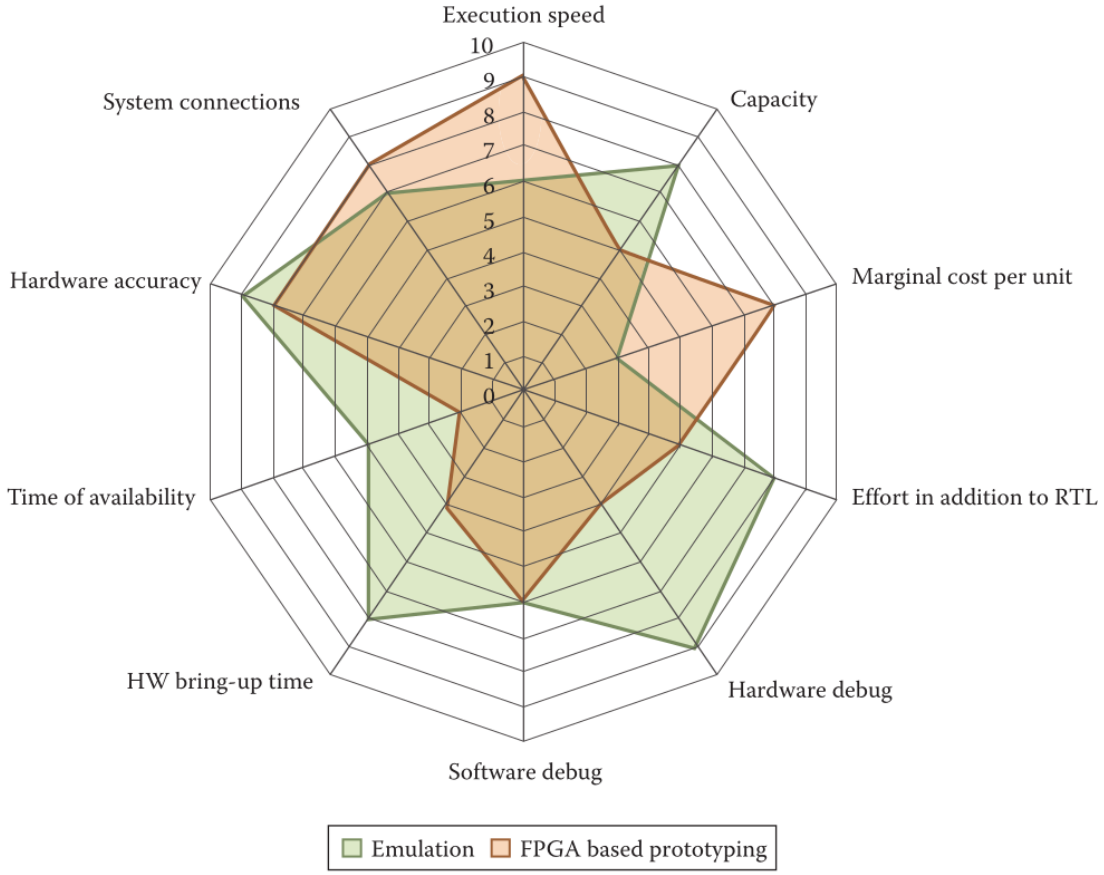
Figure 4.2: Simplified example of an emulator being used for system verification. The emulator, unlike an FPGA-based prototype board, is connected via a high-speed interface.. SOURCE: (SCHIRRMEISTER; BERSHTEYN; TURNER, 2017)



Another feature that distinguishes FPGA-based prototyping from emulators is where the DUT is configured. As emulators are detached from the target system to be verified, they are architectures designed with additional resources for hardware debugging. Emulators have more advanced logic analyzers with greater storage capacity. Thus, emulators can trace hardware bugs more easily. In FPGA-based prototypes, the trace memory is limited to the memory resources available on the FPGA itself. Therefore, the memory window may not be sufficient to identify the

root cause of a hardware bug. Figure 4.3 shows several differences between emulation and FPGA-based prototyping.

Figure 4.3: Comparison between emulation and FPGA-based prototyping. Note that emulators perform better than FPGAs in hardware debugging. However, when hardware debugging is no longer as frequent in the project, prototyping becomes more advantageous due to its execution speed for debugging software. SOURCE: (SCHIRRMEISTER; BERSHTEYN; TURNER, 2017)



In summary, emulators are excellent for debugging hardware compared to FPGA-based prototyping. Also, they are easier to use because the time spent on hardware bring-up is much shorter.

4.4.2 Advantages of emulators in education compared to prototyping

Emulators have an attractive characteristic for practical labs on digital circuits: their ability to interact with real traffic (MACMILLEN et al., 2000). Assuming the cost and complexity of emulators are not considered, they would become an attractive option for educational labs in digital circuit courses. Emulators can interface with real electronic devices, just like FPGAs in educational prototyping

boards. Moreover, as seen in Section 4.4.2, emulators

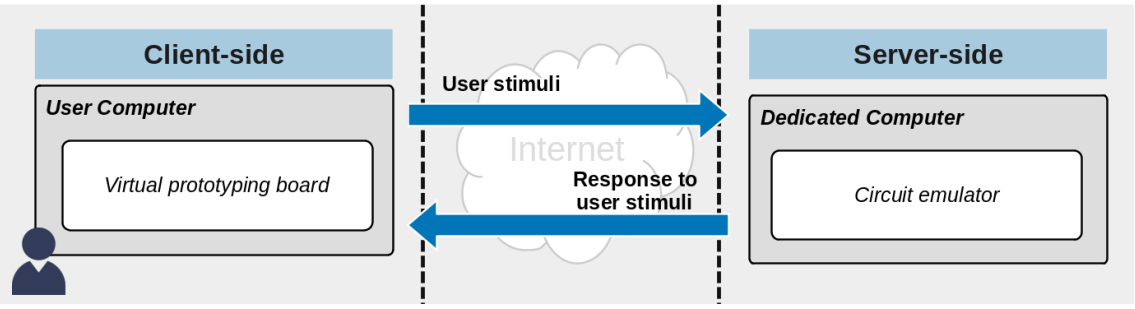
1. can be reused in other systems, meaning that the same emulator can be used in different and less expensive educational boards;
2. have the advantage over FPGAs in hardware debugging and bring-up, which is beneficial when learning digital circuits;
3. provide students with the industry best practices, preparing them for the industry reality.

All of these advantages are especially valuable in educational contexts, where students may be less experienced with hardware design and more prone to errors. Finally, note that the items above are in line with the acceptance criteria of educational technologies identified by Froyd et al., which are low cost, ease of use, and alignment with industry standards (FROYD; WANKAT; SMITH, 2012). Items 1, 2, and 3 meet these criteria, respectively.

4.5 The Pitanga emulation-based remote laboratory

This thesis proposes a remote laboratory based on the Pitanga architecture, founded on the emulator concept, where the processor or programmable device is separated from the target system. In the specific case of the Pitanga architecture, the connection interface between the target system and the emulator is the Internet. Figure 4.4 presents the general concept of the Pitanga architecture. The user experiments with the digital circuit on the left side of the figure (the client-side), while the emulator runs on the right side of the figure (the server-side).

Figure 4.4: Overview of the Pitanga remote laboratory client-server architecture. The emulator on the server-side connects to the virtual board through the Internet.

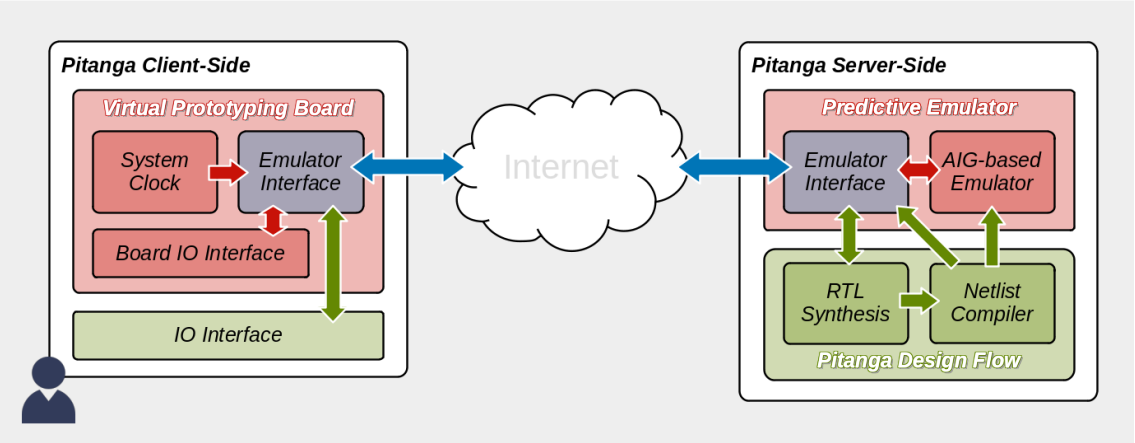


The idea is similar to FPGA-based remote laboratories, but in this case, the educational board is virtual and on the client-side with the user. The board

connects to the remote emulator through the internet, which runs on an x86 architecture processor. Therefore, the DUT (Device Under Test) runs on a cheap and highly scalable x86 general-purpose architecture offered by several cloud computing services.

Figure 4.5 shows the functional block diagram for the Pitanga platform architecture. Each block targets specific requirements on Table 1.1. As the platform intends to provide an alternative for remote digital circuit experiments, it is necessary to address the limitations imposed by FPGA-based, simulation-based, and emulation-based remote laboratories.

Figure 4.5: Functional block diagram for the Pitanga remote laboratory architecture.



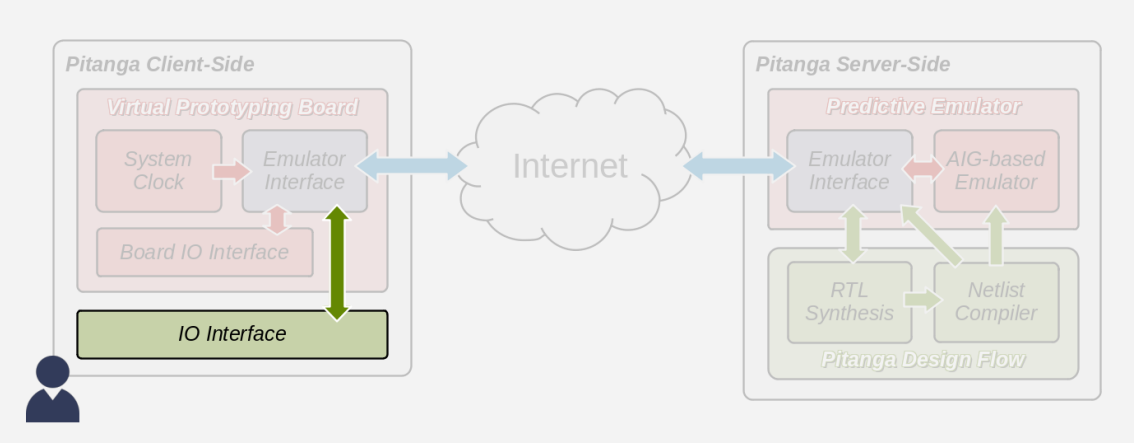
The following sections present the proposed system and detail the functional blocks comprising the Pitanga architecture's main characteristics. These blocks will be briefly introduced before delving into the details of the AIG-based emulator on the server-side.

4.5.1 IO Interface

This section describes the role of the IO interface as a component of the Pitanga remote laboratory architecture. Keep in mind that the overall architecture was presented in Figure 4.5. For increased readability, the IO interface described in this section is highlighted in Figure 4.6. In the following, we discuss the role of the IO interface.

In order to configure a digital design to run into a specific board, two types of information are necessary. The first information is the description of the digital design itself, we assume here that it is described as one or more Verilog files. The

Figure 4.6: Pitanga IO Interface as a component of the Pitanga remote laboratory architecture.



second information is the description of how the elements of the board should be connected to the inputs and outputs of the digital circuit design. This description is normally done in some proprietary format, in the case of Pitanga virtual board, pinout files are used for this purpose.

The IO interface reads the Verilog and the pinout files describing the target design, encoding them for communication with the remote emulator. The IO interface processes the design files in the Client-Side of the architecture. These design files go through the emulator interface and are sent to the Pitanga tool flow on the server-side of the architecture, which includes an RTL synthesizer and a compiler. The flow generates the main data structure for the AIG-based emulator, returning a report containing the logic gates used in the design. The compilation report is displayed by the IO interface on the client-side user screen of the Pitanga platform, as shown in Listing 4.1.

In short, the IO interface receives the design files from the user on the client-side and it sends these files for processing on the server-side. If the compilation of the design is successful, the IO interface exhibits the compilation report (or the compilation error messages if compilation is unsuccessful). Finally, when compilation is successful, the IO interface will exhibit the design configured in a virtual prototyping board as described in the next section.

Listing 4.1: Design summary report generated by the Pitanga plataform after mapping the input pinout and verilog files to the AIG-based emulator.

```
                          DESIGN SUMMARY REPORT
  module     : counter
  design file: counter.v
  pinout file: counter.pinout

Total number of wires: 95
Total number of cells: 86
Total number of ports: 23

  Cell          Instances   Cell          Instances   Cell          Instances
 ------------------------------------------------------------------------------
  AND2                  3 | NAND2                17 | XOR2                  0
  AND3                  2 | NAND3                 1 | XOR3                  0
  AND4                  0 | NAND4                 0 | XOR4                  0
  OR2                   1 | NOR2                 34 | XNOR2                 7
  OR3                   0 | NOR3                 11 | XNOR3                 0
  OR4                   0 | NOR4                  1 | XNOR4                 0
 ------------------------------------------------------------------------------
  BUF                   0 | INV                   5 | DFFRSE                4

Cells utilization: 86/500 cells (17.20 %).
```

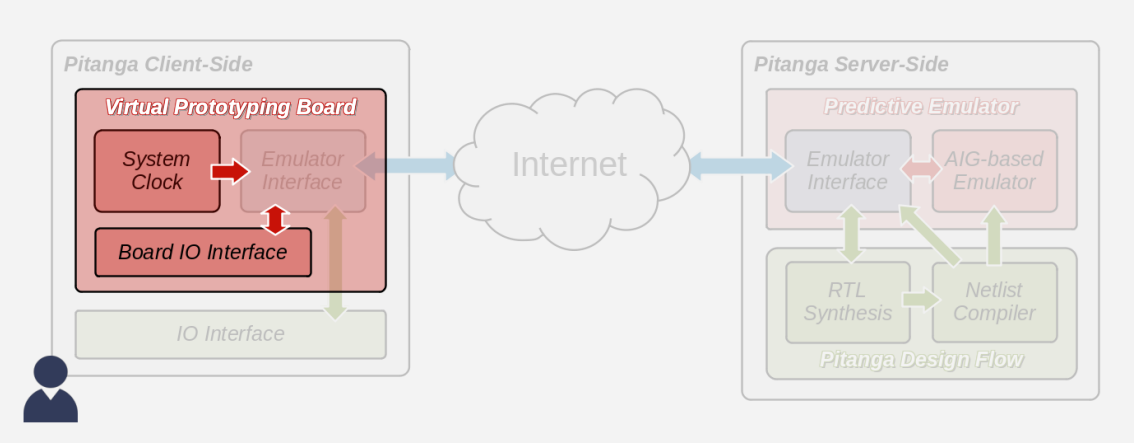
4.5.2 Virtual Prototyping Board

This section describes the role of the virtual prototyping board (COSTA; DROVES; REIS, 2023b) as a component of the Pitanga remote laboratory architecture. Keep in mind that the overall architecture was presented in Figure 4.5. For increased readability, the virtual prototyping board described in this section is highlighted in Figure 4.7.

The virtual prototyping board allows the designer to interact with the design that has been configured in the Pitanga board architecture. It comprises three blocks: the board IO interface, the system clock, and the emulator interface. Notice that emulator interface is also inside the predictive emulator block. For this reason, it will be discussed separately in Section 4.5.3.

The Board IO Interface block includes the input and output devices of the board: three push-buttons, ten switches, ten LEDs, and four seven-segment displays. The input devices, like the board itself, are virtual. Figure 4.8 shows the virtual board and its virtual IO devices.

Figure 4.7: The Pitanga virtual prototyping board as a component of the Pitanga remote laboratory architecture.



The input devices are responsive to user mouse clicks. Each mouse click on an input device generates an event that captures the state of all inputs on the virtual board. The emulator interface receives the states for further processing. The system clock is also considered an input, but instead of being controlled by the user, it is controlled by the system. The system clock generates a square wave of configurable frequency, alternating between 0 and 1. The state alternation triggers an event that captures all inputs of the virtual board and sends them to the emulator interface. Therefore, the emulator interface block receives the state of all inputs at each user or system event.

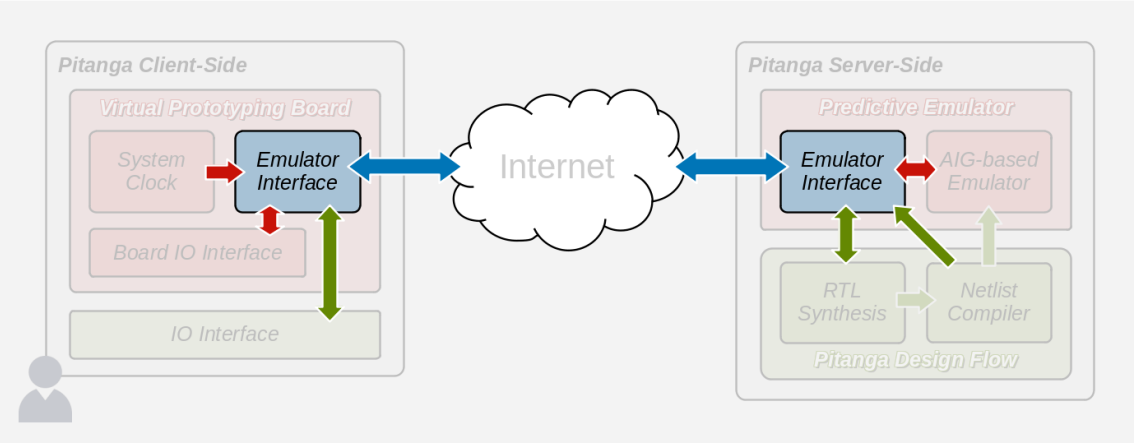
Figure 4.8: The Pitanga virtual board emulating a 16-bit accumulator. The push-buttons and toggle switches are sensitive to mouse clicks.

4.5.3 Emulator Interface

This section describes the role of the emulator interface as a component of the Pitanga remote laboratory architecture. Keep in mind that the overall architecture was presented in Figure 4.5. For increased readability, the emulator interface described in this section is highlighted in Figure 4.9. In the following, we discuss the role of the emulator interface.

As shown in Figure 4.5, the emulator interface is a block that is present at the client-side and server-side, connecting both sides. When the Pitanga virtual board is configured and the circuit is operating, the emulator interface is responsible for the communication between the client-side and the server-side. This communication entails two tasks. The first task is capturing the input signal changes made by the user through the board IO interface as well as the input changes made automatically by the system clock (in the input clock signal). These input changes are communicated from the emulator interface on the client-side to the server-side emulator interface, which then requests the AIG-based emulator to produce the corresponding new output signals. These new output signals are then sent back from the emulator interface server side to client side to be exhibited by the board IO interface.

Figure 4.9: The emulator interface as a component of the Pitanga remote laboratory architecture. The emulator interface connects to every block of the Pitanga platform.



The emulator interface is similar to a cable connecting the target system (i.e., the virtual prototyping board on the client-side) to the emulation unit (i.e., the AIG-based emulator on the server-side). However, in the Pitanga platform, this connection is not a single physical cable. Instead, the Pitanga platform uses proprietary application protocols on top of Internet standard protocols, such as the

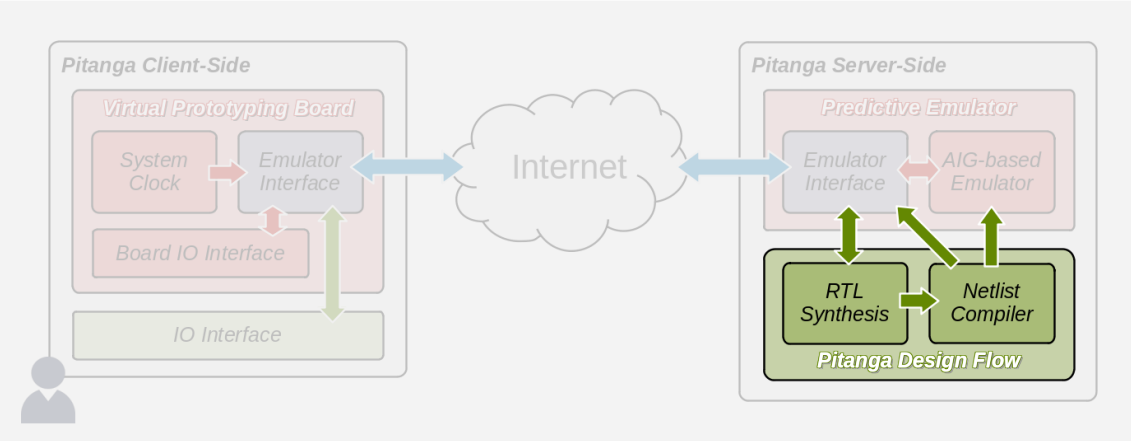
TCP/IP stack. Obviously this produces a time overhead that would increase the latency between a user acting on a switch in a virtual board and obtaining the corresponding answer from the server-side to be exhibited in the board. This is mitigated by using predictive simulation to pre-compute answers for every possible future action on the client side using a predictive emulator as described in Section 4.5.5.

The emulator interface is connected to all components in the Pitanga remote laboratory architecture. Besides the role played during circuit emulation, the emulator plays a role in circuit configuration before the emulation starts. In this sense the board is configured by a user using the IO interface to remotely access the Pitanga Design Flow component through the emulator interface. In the next section we will discuss the Pitanga Design Flow.

4.5.4 Pitanga Design Flow

This section describes the role of the Pitanga Design Flow as a component of the Pitanga remote laboratory architecture. Keep in mind that the overall architecture was presented in Figure 4.5. For increased readability, the Pitanga Design Flow described in this section is highlighted in Figure 4.10. In the following, we discuss the role of the Pitanga Design Flow.

Figure 4.10: Overview of the Pitanga design flow as a component of the Pitanga remote laboratory architecture.

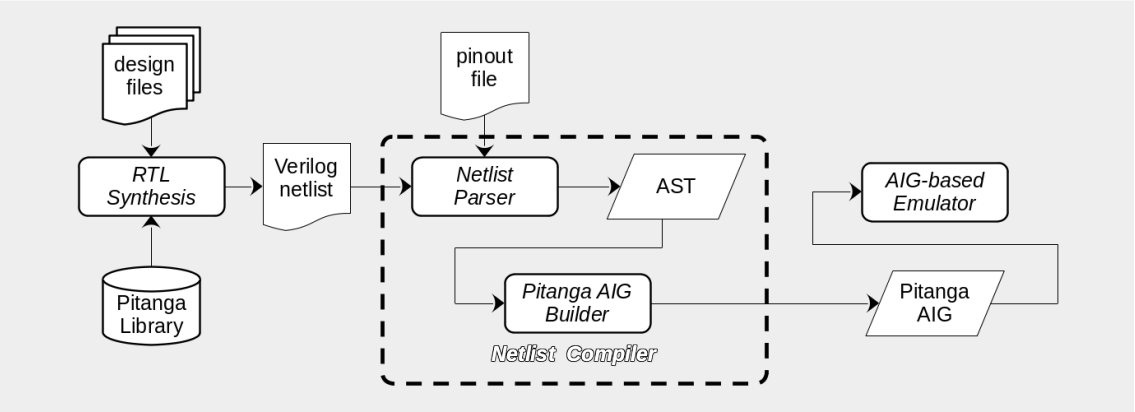


The Pitanga design flow is used to convert the design files into an AIG file that will be used in circuit emulation. The Pitanga design flow consists of the RTL Synthesis and Netlist Compiler blocks. Figure 4.11 overviews the Pitanga design

flow.

The RTL synthesis block uses the open-source YOSYS/ABC tool (Claire Xenia Wolf, 2023; BRAYTON; MISHCHENKO, 2010), which reads the input Verilog project files and the Pitanga virtual cell library. The library is described in Liberty format, is located on the server-side, and cannot be modified by the user. The YOSYS/ABC tool synthesizes the project files and delivers a Verilog netlist mapped to the Pitanga cell library. The Netlist Compiler reads the mapped verilog netlist and produces the AIG for emulation.

Figure 4.11: Overview of the Pitanga design flow components.



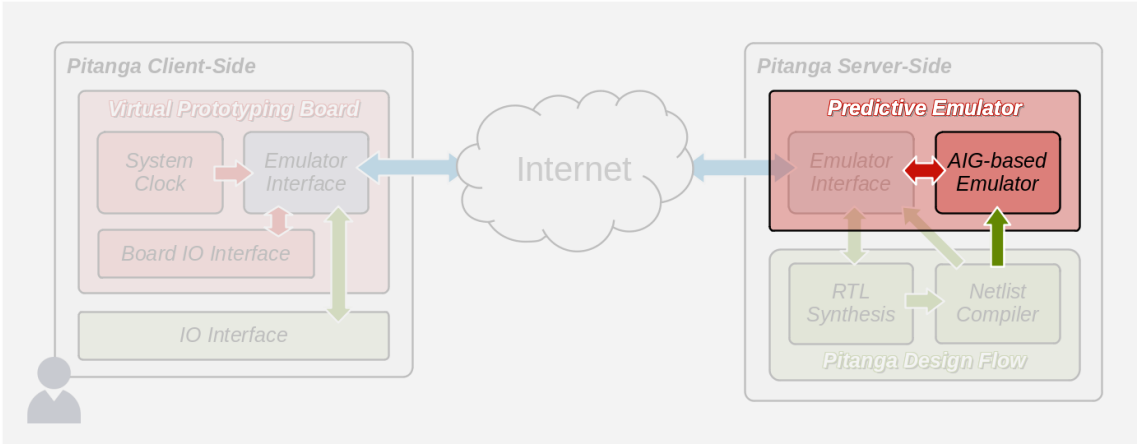
The Netlist Compiler parses the Verilog netlist and the pinout file, creating an abstract syntax tree (AST) (AHO et al., 2007). From the AST, the data structure capable of computing sequential digital circuits, called Pitanga AIG, is built. The Pitanga AIG is made based on the structure presented in Section 2.2.1 but modified to support the computation of sequential circuits using flip-flops. The AIG is used for predictive emulation as described in the next section.

4.5.5 Predictive Emulator

This section describes the role of the Predictive Emulator as a component of the Pitanga remote laboratory architecture. Keep in mind that the overall architecture was presented in Figure 4.5. Figure 4.12 highlights the Predictive Emulator (COSTA; DROVES; REIS, 2023d) described in this section for improved readability. In the following, we discuss the role of the Predictive Emulator.

The predictive emulator is the core of the Pitanga platform. In this section, we will explain the data structures and the algorithm used in the AIG-based emu-

Figure 4.12: AIG-based emulator as a component of the Predictive Emulator.



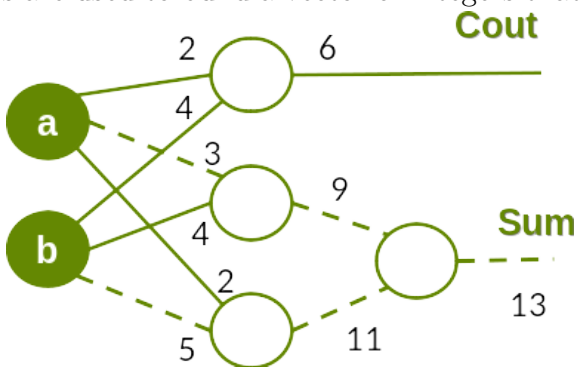
lator. The data structure used in the AIG-based emulator is an and-inverter graph (AIG). Every node in an AIG is a two-input AND logic gate, except for the primary input nodes of the AIG. Each node is associated with a pair of integers in the format $\{n, n+1\}$, where n is a positive integer and $n+1$ is the subsequent odd positive integer.

In the AIGs of Figures 4.13 and 4.14, solid lines in the arcs represent that the node output is a direct signal (even number), while dashed lines in the arcs represent that the signal is in the negated form (odd number). Therefore, the integer used to identify the node also defines the node output signal polarity (i.e., direct or negated signal). The half-adder represented as an AIG in Figure 4.13 shows that.

The half-adder has inputs a and b and outputs Sum and $Cout$. These are external labels for readability; internally, each label is represented by a pair of numbers. For instance, the pair of numbers {2,3} is used to reference input a. Similarly, the pair of numbers {4,5} is used to reference input b. The label a is a variable, and the pair of numbers {2,3} is used to represent literals of variable a, such that $a = 2$ and $!a = 3$. This way, it is possible to refer to both literals of a given variable as numbers. This representation allows storing an AIG as a table of vectors of integers, as described in the following.

Figure 4.14 shows the AIG representation for a half-adder as a table of vectors of integers. The leftmost column indexes the AIG node and its input values $i0$ and $i1$. The index of the table is the node (even) identifier divided by two. For instance, node $a = 2$, so the index to store node a is 1. Similarly, node $b = 4$, so the index to store node b is 2. As inputs to the circuit have no inputs to themselves, columns $i0$ and $i1$ repeat the node identifier. So, for a the vector index 1 indicates

Figure 4.13: A half-adder implemented in the Pitanga platform as an AIG. The numbers on the edges are used to build a vector of integers that represents the AIG.



$a = i0 = i1 = 2$. Similarly, for b the vector index 2 indicates $b = i0 = i1 = 4$.

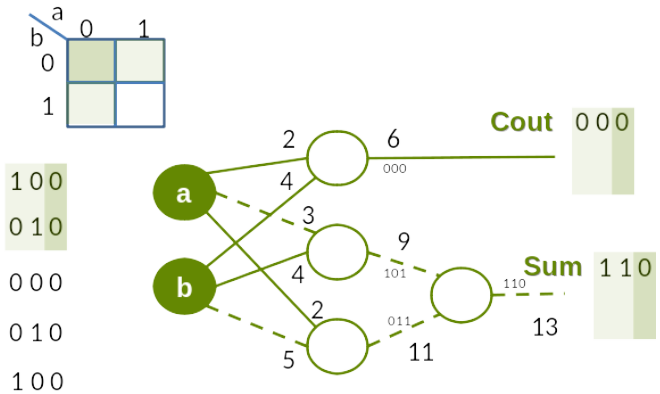
The output $Cout$ has the number 6, so it is stored in index 3. We recall that the internal nodes are two-input AND logic gates, so index 3 contains inputs i0=2 and i1=4, indicating that output $Cout = (a) \cdot (b)$, or numerically: $6 = (2) \cdot (4)$. The output Sum has the number 13, so it is stored in index 6. We recall that the number 13 is paired with 12, so they share the index 6 as $12/2 = 6$. Notice that index 6 contains i0=9 and i1=11, indicating that output Sum =!$((9) \cdot (11))$, or numerically: 13 =!$((9) \cdot (11))$, or yet !12 =!$((!8) \cdot (!10))$.

The AIG in Figure 4.14 has two intermediate nodes with numbering $\{8, 9\}$ and $\{10, 11\}$. Numerically, index 4 from the vector states that $8 = 3 \cdot 4$, while index 5 expresses that $10 = 2 \cdot 5$. **This way, the AIG representation can be described as a table of vectors of integers** as depicted in Figure 4.14.

Figure 4.14: An AIG representation for a half-adder. The table on the left describes the AIG using the vector of integers $i0$ and $i1$.

Predictive Inputs
Inputs: a=0; b=1; → Outputs: Cout=0; Sum=1;
Inputs: a=1; b=0; → Outputs: Cout=0; Sum=1;

Index	i0	i1
1	2	2
2	4	4
3	2	4
4	3	4
5	2	5
6	9	11



Once the data structure has been understood, it is possible to understand the predictive emulator algorithm. The algorithm will be explained for the half-adder

in Figure 4.14. Firstly, consider that the board starts with inputs $ab = 00$ (current value, or c for short). The Karnaugh map in Figure 4.14 highlights the inputs $ab = 00$ in green. As there are two inputs, there are also two vectors with hamming distance 1 from $ab = 00$: $ab = 10$ (hamming a, or ha) and $ab = 01$ (hamming b or hb). Considering the vectors ha, hb, and c represent possible future states of the inputs of the virtual prototyping board, it is possible to organize these values as a triplet of future input states F_s. Thus, we have:

$$F_s = \{ha, hb, c\} = \{\{10\}, \{01\}, \{00\}\} \tag{4.1}$$

The triplet of future input states F_s described in equation 4.1 is a list of vectors. This list of vectors can also be viewed as a matrix of 3 lines and 2 columns, obtaining

$$F_s = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{4.2}$$

By transposing equation F_s in Equation 4.2, we have:

$${F_s}^{\mathrm{T}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.3}$$

Notice that each line of the matrix 4.3 now represents the future input states of variables a and b, respectively (the rightmost value represent the future state with no input changes). For variable a, the line is $a = [1, 0, 0]$. Similarly, for variable b, the line is $b = [0, 1, 0]$. Applying a bitwise AND operation in both lines, we have

$$a \cdot b = 100 \cdot 010 = 000 \tag{4.4}$$

The Equation 4.4 results in the triplet $Cout = \{0, 0, 0\}$ from the AIG representation in Figure 4.14. The graph shows that $Cout = (6) = (2) \cdot (4)$ and these numbers can be divided by 2 and then converter to indexes for the AIG table representation. As the AIG table representation is a Directed Acyclic Graph (DAG), the values of the triplets - i.e., future states - can be propagated in the AIG using bitwise logic AND and logic NOT operations. **Therefore, the predictive emulator algorithm only does a single pass on the AIG table. The runtime is**

linear with respect to the table size, which is linear with the number of AIG nodes in the circuit being emulated.

Notice that by using bitwise logic operations and hamming distance 1 it is possible to determine the output values for every possible input. Instead of computing a single boolean value for each node in the circuit, the simulation is performed with multiple values for each node. Figure 4.14 illustrates that the triplet obtained for the output Sum in the predictive simulation is $Sum = \{1, 1, 0\}$. The current value c for Sum given by $c(Sum) = 0$, meaning that for $ab = 00$ the output $Sum = 0$. The predictive value for a change in input a is $ha(Sum) = 1$, meaning that for $ab = 10$ the output $Sum = 1$. Finally, the predictive value for a change in input b is $hb(Sum) = 1$, meaning that for $ab = 01$ the output $Sum = 1$.

The predictive emulation is performed on the server-side. However, the answers transmitted to the client-side contain the future values to be exhibited at the outputs. This way, the client already knows in advance the answers for every possible user interaction, updating the virtual prototyping board immediately as the set of possible answers is already stored on the client-side. At the same time the client updates the virtual board, the emulator interface triggers a new request to the AIG-based emulator on the server-side. The AIG-based emulator then provides a new set of answers - with hamming distance one - considering the new status.

4.6 Contributions of this chapter

This chapter presented the core of the thesis proposal. The architecture of the system was described. The description of the algorithm for predictive emulation was done with the example of a simple half-adder circuit, highlighting that the emulation runtime is linear on the number of nodes in the AIG. Also, this chapter has described how the system mitigates latency through pre-computing output values using predictive emulation with hamming distance 1.

5 METHODOLOGY

Identifying and measuring the *user tolerance* is crucial because a novice student can wrongly interpret the lack of responsiveness of the system as a failure in the DUT. As the Pitanga platform is an educational tool, such perception by the student is not desirable.

The following sections present the methodology used in this thesis to measure the Pitanga system responsiveness. Section 5.1 and 5.2 describes the input and output parameters, respectively. Section 5.3 explains the experiment environment and how the input parameters control the experiments. Then, Section 5.4 demonstrates how the output parameters are collected for further analisys.

5.1 Output parameters

A cause-and-effect relationship has a specific period between the observable cause and the observable effect. This period can be the time elapsed from clicking a mouse (the observable cause) to firing a weapon in a first-person shooter game (the observable effect). It can also be the touching of a tablet screen (the observable cause) until the rotating of a piece in a puzzle game (the observable effect). In the context of multiplayer online game development, this time delay is referred to as latency (GLAZER; MADHAV, 2015). Depending on the nature of the game, the latency duration may be more or less tolerable to the user.

Like a game, the Pitanga platform has a latency between pressing a button (the observable cause) and the lighting of an LED (the observable effect). However, instead of waiting for a switch to be pressed before beginning to calculate the answer, the Pitanga platform performs this operation in a predictive manner. By computing the solution for all possible events previously, the output devices on the virtual board (such as LEDs and seven-segment displays) are immediately updated when the user interacts with the board. As all the possible results are already stored on the client-side, the virtual prototyping board commits the solution in the output devices when an event occurs. Immediately, the board dispatches a new request to the predictive emulator on the server-side for computing the new set of possible solutions.

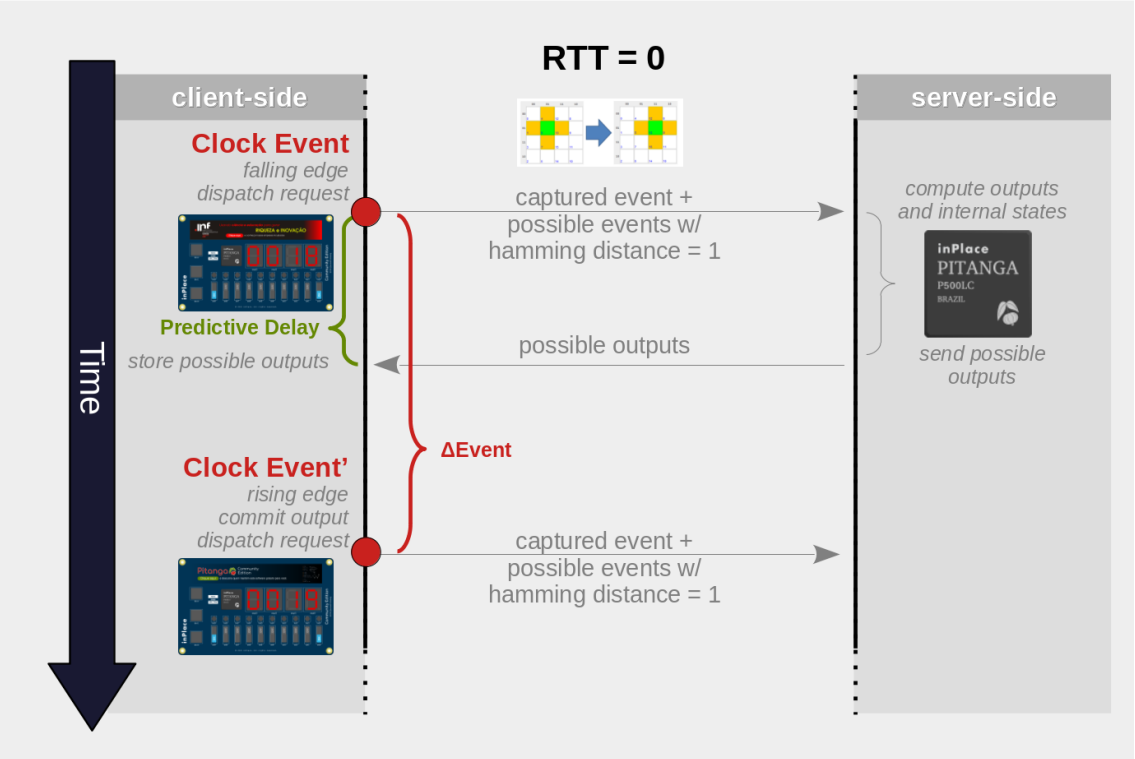
The predictive emulator eliminates latency because the observable effect is already stored on the board for every possible observable cause. However, this tech-

nique has limitations, specifically when the predictive emulator is not fast enough to deliver the answer before the next event on the board. For example, the predictive calculation algorithm for a combinational adder circuit is expected to be faster than the interval between two addition operations requested by the user. The same is expected when two consecutive clock events are faster than the predictive emulator. In such situations, latency may be noticeable by the user and even not tolerable.

5.1.1 Predictive Delay

Figure 5.1 is a temporal diagram showing the relationship between the *predictive delay* and two consecutive system clock events, denoted as $\Delta Event$. The red circles are the events related to the rising and falling edges of the *system clock*. Each edge triggers a request to the predictive emulator on the server-side, which calculates all the possible solutions of the circuit. After finishing the calculation, the emulator interface transmits the answers back to the client-side, which stores them in the virtual board. The *predictive delay* written in green in Figure 5.1 indicates the elapsed time of this operation.

Figure 5.1: Predictive delay and system clock with no network delay (RTT = 0)



The predictive delay is the output parameter that ensures the system respon-

siveness if it is smaller than $\Delta Event$. The inequation 5.1b shows this relationship as

$$Predictive\ Delay \leq Clock\ Event' - Clock\ Event \tag{5.1a}$$

$$Predictive\ Delay \leq \Delta Event \tag{5.1b}$$

However, $\Delta Event$ is not a function of two consecutive clock events ideally spanned in time. For example, the clock rising and clock falling events may vary depending on factors such as the processing load on the Pitanga client-side. In addition, the client-side awaits the incoming data from the server-side before committing the outputs to the virtual board and dispatching the next computation request. The Pitanga platform behaves in a way that the $\Delta Event$ from Figure 5.1 is always greater than the predictive delay.

5.1.2 System Tolerance

In order to differentiate the varying $\Delta Event$ from an ideally spanned $\Delta Event$, Equation 5.1b must be rewritten as

$$Predictive\ Delay \leq \Delta Event_{ideal} \tag{5.2}$$

where,

$$\Delta\ Event_{ideal} = \frac{Clock\ Period_{ideal}}{2} \tag{5.3}$$

Noticed that $\Delta Event_{ideal}$ is a constant because it depends solely on the nominal system clock from the client-side. This constant delimits the amount of time that the predictive delay affects the system responsiveness. Then, by defining that

$$System\ Tolerance = \Delta Event_{ideal} \tag{5.4}$$

and applying equation 5.4 to inequality 5.2, we obtain:

$$Predictive\ Delay \leq System\ Tolerance \tag{5.5}$$

The Inequation 5.5 delimits the amount of time that the predictive delay cannot exceed for the Pitanga platform to run as expected. The system tolerance is the parameter that defines this amount of time.

5.1.3 User Tolerance

The lack of responsiveness of the system is not noticeable to the user when the predictive delay is slightly greater than the system tolerance. The predictive delay needs a certain amount of time over the system tolerance to become evident to the user. Although there is no exact definition for this amount of time, this time exists and depends on the nature of the application.

In online multiplayer games, this time delay, or latency, can vary from up to 20ms for virtual reality games to 500ms for real-time strategy games (GLAZER; MADHAV, 2015). For the Pitanga platform, we consider a latency of up to 150ms tolerable, the same as used in first-person shooter games. Thus, we have:

$$User\ Tolerance = System\ Tolerance + 150ms \tag{5.6}$$

Since *user tolerance* is greater than the *system tolerance*, by applying Inequation 5.5 we conclude that

$$Predictive\ Delay \leq System\ Tolerance \leq User\ Tolerance \tag{5.7}$$

When true, the Inequation 5.7 means the Pitanga platform is running with high responsiveness. However, this inequality may become false depending on the input parameters, meaning that the system is unresponsive.

5.1.4 Slack

The slack is used in conjunction with Inequality 5.7. Through this parameter, it is possible to identify whether the system is responsive or not. Below is its definition:

$$Slack = System\ Tolerance - Predictive\ Delay \tag{5.8}$$

The slack parameter has a comparable definition to the slack employed in STA tools. For instance, a positive slack indicates that the timing in the Pitanga platform is functioning correctly. However, a negative slack *may imply* a latency that the user may not tolerate.

It should be emphasized that a negative slack *may imply* a latency not tolerable for the user because the slack may be negative and not perceived by the user. This scenario can occur when the predictive delay exceeds the system tolerance but not the user tolerance.

In addition to the *predictive delay*, the *system tolerance* and the *user tolerance*, the slack is the fourth output parameter employed in this study to validate the thesis that students can remotely prototype responsive digital circuits without using FPGA-based prototyping boards.

5.2 Input parameters

The experiments must collect the output parameters predictive delay, system tolerance, user tolerance, and slack. However, these parameters depend on input parameters. Knowing and controlling the input parameters is the key to effective and reproducible experiments.

5.2.1 Transistor count

The more logic cells the circuit has, the bigger the AIG in memory and, therefore, the longer the emulation processing time. The longer the emulation time, the longer the predictive delay.

As logic cells are nothing more than a group of transistors, and a digital circuit is a group of logic cells, each circuit running in the Pitanga platform can be measured by its number of transistors. The transistor count affects the predictive delay; consequently, it is one of the input parameters.

Table 5.1 details the transistor count for each cell in the Pitanga library. The number of transistors considers typical CMOS logic gate designs (WESTE; HARRIS, 2010). The only exception is the DFFRSE flip-flop which uses transmission gates in its design.

5.2.2 System frequency

The faster the elapsed time between two events, the faster the predictive emulator on the server side must process and deliver the set of possible outputs to the virtual prototyping board on the client-side.

Two consecutive events are more controllable and reproducible when they do not depend on human interference. The system clock on client-side is such an event that does not depend on human interference. In addition, the system clock can easily control the time elapsed between two events by modifying its operation frequency directly in the source code.

The experiments in this thesis use the system frequency as an input parameter to control the time elapsed between consecutive events. Modifications on this parameter affect the system tolerance and, consequently, the user tolerance.

5.3 Running the experiments

The experiments require a prepared and configured environment in order to be replicated. Section 5.3.1 defines two configurations for evaluating the Pitanga platform, Section 5.3.2 details the hardware specification for the predictive emulator on the server-side, and Section 5.3.3 shows how the input circuit and design flow is controlled.

5.3.1 Preparing the environment

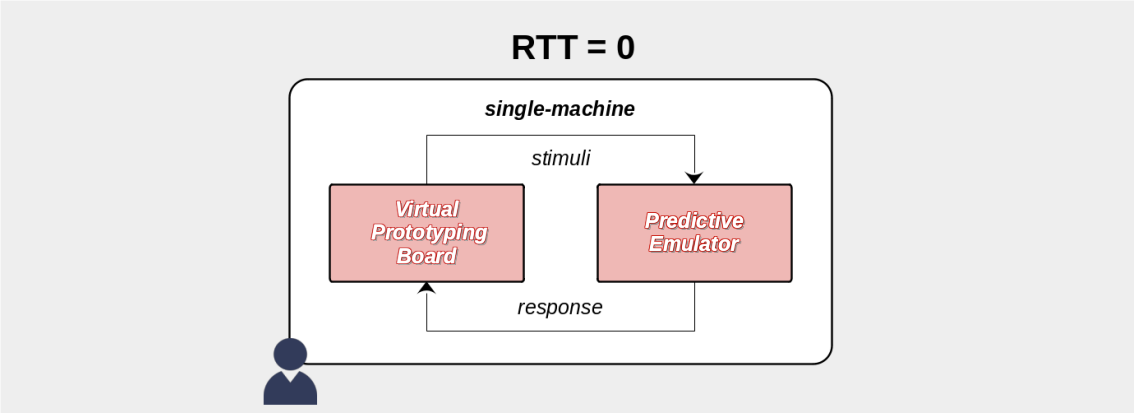
This work performs two sets of experiments. The first set runs the client-server architecture on a single machine (i.e., both client-side and server-side running

Table 5.1: Virtual transistor count for each cell in the Pitanga library.

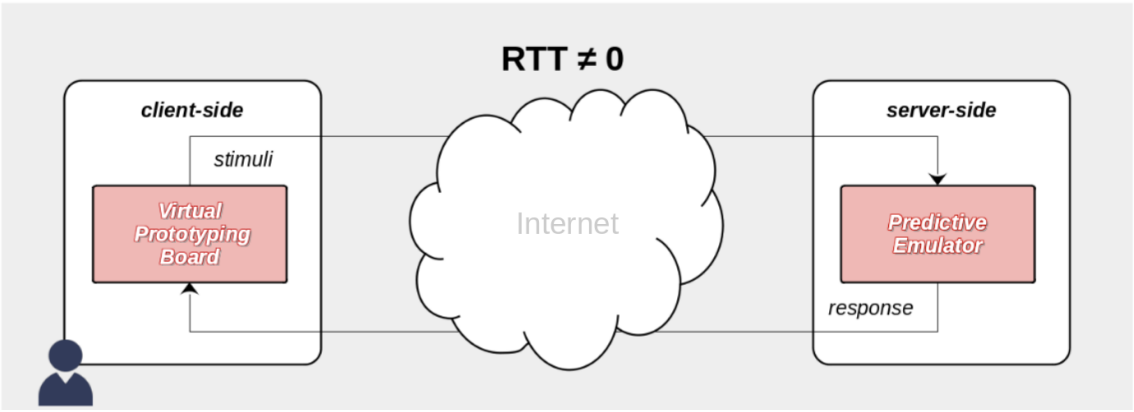
Cell	Transistors	Cell	Transistors	Cell	Transistors
AND2	6	NAND2	4	XOR2	12
AND3	8	NAND3	6	XOR3	22
AND4	10	NAND4	8	XOR4	32
OR2	6	NOR2	4	XNOR2	12
OR3	8	NOR3	6	XNOR3	22
OR4	10	NOR4	8	XNOR4	32
BUF	4	INV	2	DFFRSE	38

concurrently on the same machine). The second set runs the same experiments on a client-server architecture of separate machines. Figure 5.2 shows the concept of the experiment environment for the single-machine and client-server architecture, respectively.

Figure 5.2: Different configurations for evaluating system and user tolerance



(a) Single-machine architecture

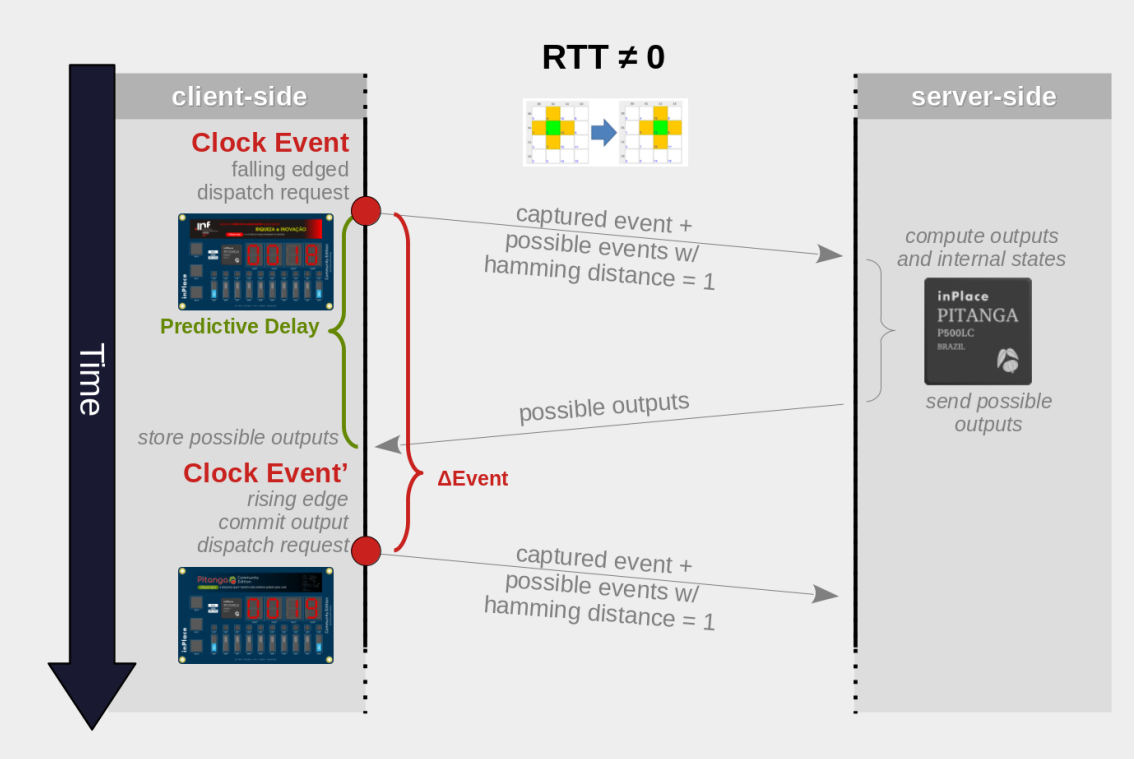


(b) Traditional client-server architecture

Notice that the single-machine architecture in Figure 5.2 (a) is equivalent to the timing diagram indicated in Figure 5.1. On the other hand, the architecture shown in Figure 5.2 (b) is equivalent to the timing diagram in Figure 5.3.

The single-machine experiment provides a more dynamic, cost-effective, and streamlined environment for validating changes compared to the traditional client-server architecture that requires configuring an extra remote computer. Besides, the single-machine architecture may also serve as an alternative application.

Figure 5.3: Predictive delay and system clock with network delay (RTT ≠ 0)



5.3.2 Hardware specs

The Pitanga Platform was designed to run on low-end CPUs to meet the needs of students and educational institutions that cannot afford to acquire high-end computers or FPGA-based prototyping boards (requirement 4, Table 1.1). For this reason, the experiments on the client-side run on a 2010 computer. Listing 5.1 details the specification of this computer.

Listing 5.1: Target architecture for evaluating test on the client-side.

```
Notebook DELL Inspiron 14 N4050
CPU: Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz Quad-Core
RAM: DDR3 2x4 GB 1333 MHz
HDD: SSD WDC WDS240G2G0A- 240 GB
```

When running on a remote machine, the server-side runs on a Google Cloud e2-micro server (CLOUD, 2023). Listing 5.2 details the specification of the e2-micro machine.

Listing 5.2: Target architecture for evaluating test on the client-side.

```
Machine: e2-micro
vCPUs: 2
```

```
Memory (GB): 1
Local SSD: no
```

5.3.3 Controlling the experiments

The input parameters, as the experimenting environment, must be controllable in order to be reproducible. Therefore, the experiments must:

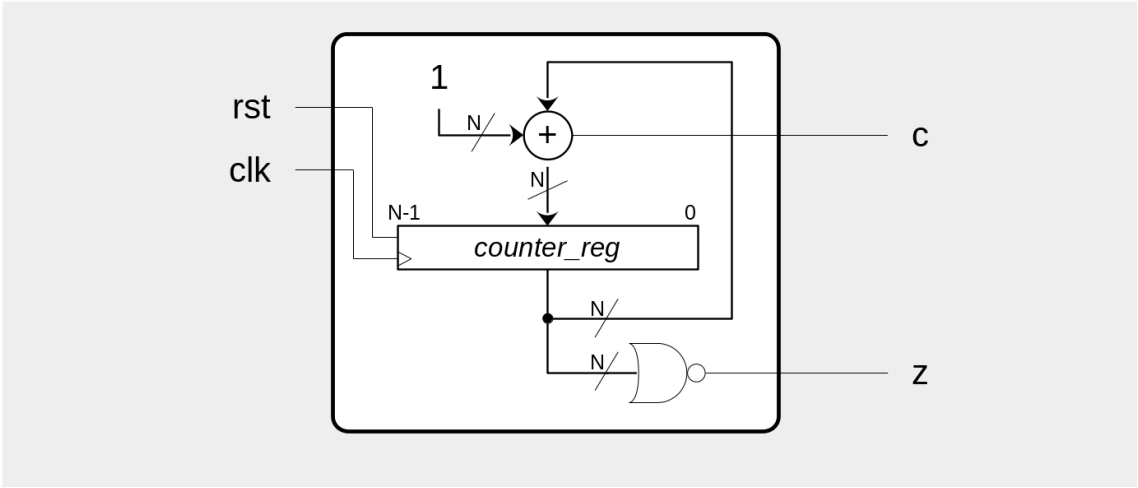
1. easily allow the modification of the transistor count in the input design;
2. run each Verilog netlist on different clock frequencies;

The number of transistors can be easily modified and reproducible with the proper input circuit design. Counters with different bit widths satisfy these criteria because they:

1. can be parameterized in Verilog, allowing the execution of counters of different sizes.
2. are sequential circuits dependent on clock events and can therefore be emulated at different frequencies on the Pitanga platform;

Figure 5.4 shows the schematic for an N-bit counter. The counter output bits connect to a bit-wise NOR gate that sets the z output flag whenever the counter equals zero. The carry out signal from the N-bit adder also connects to the c output of the module. So, regardless of the size of the counter, the input and output signals of the module remain constant.

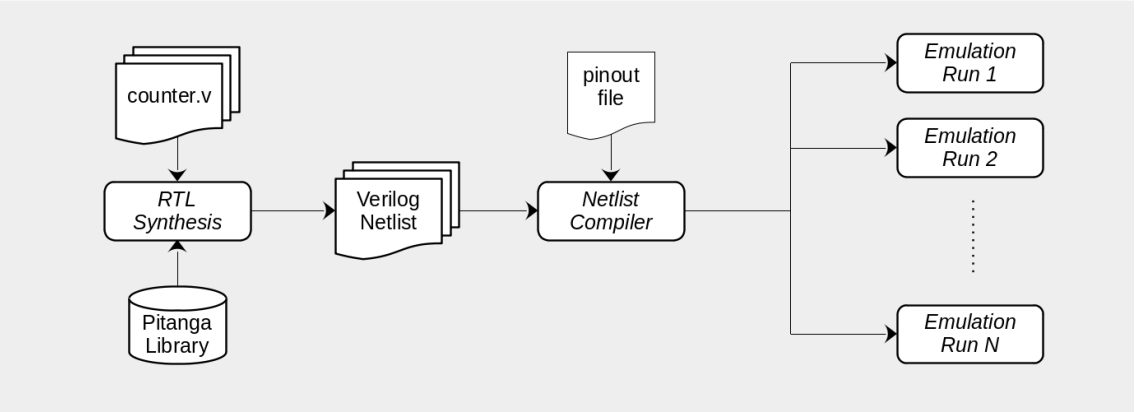
Figure 5.4: Schematic for a N-bit counter



A fixed number of IOs enables the usage of the same pinout file for N-bit

counters of different sizes, enabling the automation of the experiments. Figure 5.5 shows the design flow for the experiments. Notice that each emulation run has an associated N-bit counter netlist.

Figure 5.5: Automated design flow for the experiments



Eight counters of sizes 8, 16, 32, 64, 128, 256, 512, and 1024 bits generate de data input stimuli. The Yosys tool synthesizes each counter and maps them to separated Verilog netlists compatible with the Pitanga netlist compiler. Yosys uses the Pitanga cell library for mapping the N-bit counters. This library comprises the cells available in the Pitanga emulator as shown in Table 5.1. As the transistor count is directly related to the number and type of cells, each experiment determines a specified number of transistors.

The experiments aim to collect and calculate the output parameters *predictive delay*, *system tolerance*, *user tolerance*, and *slack* for different-sized counter circuits. Each experiment runs at a specific clock frequency, starting at 1Hz. The other frequencies are linearly spaced at every 6 Hertz, i.e., 1Hz, 7Hz, 13Hz, 19Hz, and 25Hz. The frequency spacing is linear because it helps to identify the experiments that the *slack* begins to affect the system responsiveness. The maximum frequency is 25Hz because the scheduler of the development framework is limited to 60 frames of event per second (Kivy, 2023).

Notice that any frequency above 30 Hz will place two or more events in the same event frame of the scheduler. In other words, frequencies above 30Hz trigger two or more predictive emulations; however, only one run has temporal validity. Therefore, operating the virtual board at frequencies higher than 30Hz is ineffective because it consumes the CPU unnecessarily.

5.4 Collecting the samples

The input parameters system frequency and transistor count control the experiments. There are five input system frequencies and eight different N-bit counter designs. However, as shown in Figure 5.2, there are two different environment configurations. So, the methodology collects samples for the $8 \cdot 5 \cdot 2 = 80$ experiments.

Each experiment collects 200 samples. The events that trigger the collection of samples are the clock edges and the data arrivals on the client-side. Just after an event triggers, the experiment environment samples the experiment name, the system frequency, the sample id, the event type, and the timestamp. The sample is saved into a CSV file as shown in listing 5.3.

Listing 5.3: Measurements collected in a sample of the experiments.

```
EXPERIMENT,FREQUENCY,SAMPLE_ID,EVENT_TYPE,TIMESTAMP
counter8b,1,1,build_start,1675774117,74082
counter8b,1,2,build_finish,1675774118,4461
counter8b,1,3,rising_edge,1675774118,53675
counter8b,1,4,data_arrival,1675774118,56587
counter8b,1,5,falling_edge,1675774119,02927
counter8b,1,6,data_arrival,1675774119,05841
counter8b,1,7,rising_edge,1675774119,53152
counter8b,1,8,data_arrival,1675774119,57588
counter8b,1,9,falling_edge,1675774120,03678
counter8b,1,10,data_arrival,1675774120,06534
```

The sample is composed of five columns, each of them containing data. Each collum is explained as follows:

- EXPERIMENT: experiment name identifying the counter width;
- FREQUENCY: frequency of the experiment system clock in Hertz;
- SAMPLE_ID: identification of the sample in the current experiment;
- EVENT_TYPE: type of the event. They are *build_start*, *build_finish*, *rising_edge*, *data_arrival*, and *falling_edge*.
- TIMESTAMP: timestamp in seconds of the sample;

The compilation time is associated with the event type *build_start* and *build_finish*. This work does not detail these events because they are out of the

scope of the thesis.

5.4.1 Computing the predictive delay

The experiment environment processes the CSV file after collecting the samples for every experiment. The predictive delay related to rising and falling clock edges are obtained by computing the difference of the timestamp as follows:

$$Predictive\ Delay_r = TIMESTAMP_{data\_arrival} - TIMESTAMP_r \tag{5.9a}$$

$$Predictive\ Delay_f = TIMESTAMP_{data\_arrival} - TIMESTAMP_f \tag{5.9b}$$

5.4.2 Computing the tolerances

The system and user tolerance are constant values as defined by Equations 5.4 and 5.6, respectively. The experiment environment computes these parameters once and repeats their value for each sample.

5.4.3 Computing the slack

By applying 5.9a and 5.9b on equation 5.8, we obtain one slack related to the rising edge and another slack related to the falling edge. These equations are as follows:

$$Slack_r = System\ Tolerance - Predictive\ Delay_r \tag{5.10a}$$

$$Slack_f = System\ Tolerance - Predictive\ Delay_f \tag{5.10b}$$

Although equations 5.10a and 5.10b produce different results, the differences are not significant for the analysis.

5.4.4 Computing the transistor count

For each experiment, the Pitanga platform generates a design summary containing the number of equivalent CMOS transistors of the design. These values are necessary to compare the complexity of projects emulated on the Pitanga platform with real IC designs, such as those shown in the Figure 2.1.

5.5 Contributions of this chapter

This chapter presented the methodology used to evaluate the responsiveness of the Pitanga platform. It has defined the input and output parameters, the experiment environment, and how the input parameters control the experiments in order to be reproducible. It also explained how the output parameters are collected and stored for further analysis.

6 RESULTS

This chapter presents the effectiveness of the predictive emulation algorithm explained in Section 4.5.5. The chapter presents several charts using the results obtained through the methodology presented in Chapter 5.

Section 6.1 shows the linear complexity of the developed algorithm. Section 6.2 shows a series of charts demonstrating the average predictive delay as a function of the system frequency and the number of transistors. Operating ranges for the Pitanga platform are identified in Section 6.3, revealing that the frequency and the project size are closely related to the system responsiveness decay. Section 6.4 demonstrates the emulation capacity of the Pitanga platform by comparing sample designs with real projects of historical CPUs. Finally, Section 6.5 presents the contributions of this chapter.

6.1 Time Complexity

Figure 6.1 illustrates the emulation runtime for N-bit counters of 8, 16, 32, 64, 128, 256, 512, and 1024 bits. Each counter has a specific number of associated transistors: 494, 1040, 2166, 4414, 8884, 18052, 36622, and 72486, respectively. The Pitanga platform computes the total number of transistors according to the number of transistors in each counter cell. The number of transistors in each cell is obtained from the design summary and the values in Table 5.1.

Note that the emulation runtime chart is directly proportional to the number of transistors. Since the number of transistors is related to the type of cell, and each cell is associated with AIG nodes, we have the following:

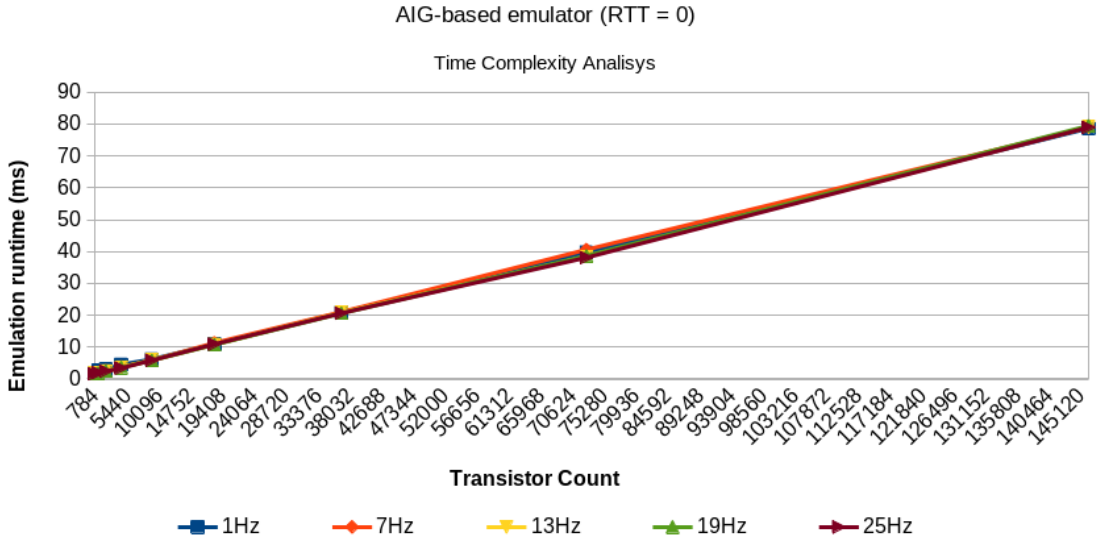
$$number\ of AIG\ nodes \propto number\ cells \propto number\ of\ transistors \tag{6.1a}$$

$$number\ of AIG\ nodes \propto number\ of\ transistors \tag{6.1b}$$

The chart shown in Figure 6.1 demonstrates that the emulation runtime on the server-side is linearly proportional to the number of AIG nodes in the emulated circuit. In other words, the AIG-based emulator takes $O(n)$ time to calculate every possible circuit output, as discussed in Section 4.5.5. Thus, as derived from the

proportionality described in 6.1b, the emulation runtime is also proportional to the expected number of transistors in the circuit.

Figure 6.1: Time complexity analysis for the AIG-based emulator (RTT=0).



It is important to point out that the emulation runtime samples collected in Figure 6.1 are the minimum values from a population of 102 predictive emulator runs in the single-machine architecture (RTT=0). Considering the average value for the traditional client-server architecture (RTT≠0), the chart would have deviations due to network latency.

6.2 Predictive Delay Analysis

The predictive delay is a linear function proportional to the number of transistors in the emulated circuit. The predictive delay has the same meaning as the emulator runtime. The following section uses predictive delay instead of emulator runtime

In Sections 6.2.1 and 6.2.2, we present the predictive delay for different N-bit counter, indicating when the delays begin to impact system responsiveness.

6.2.1 RTT=0

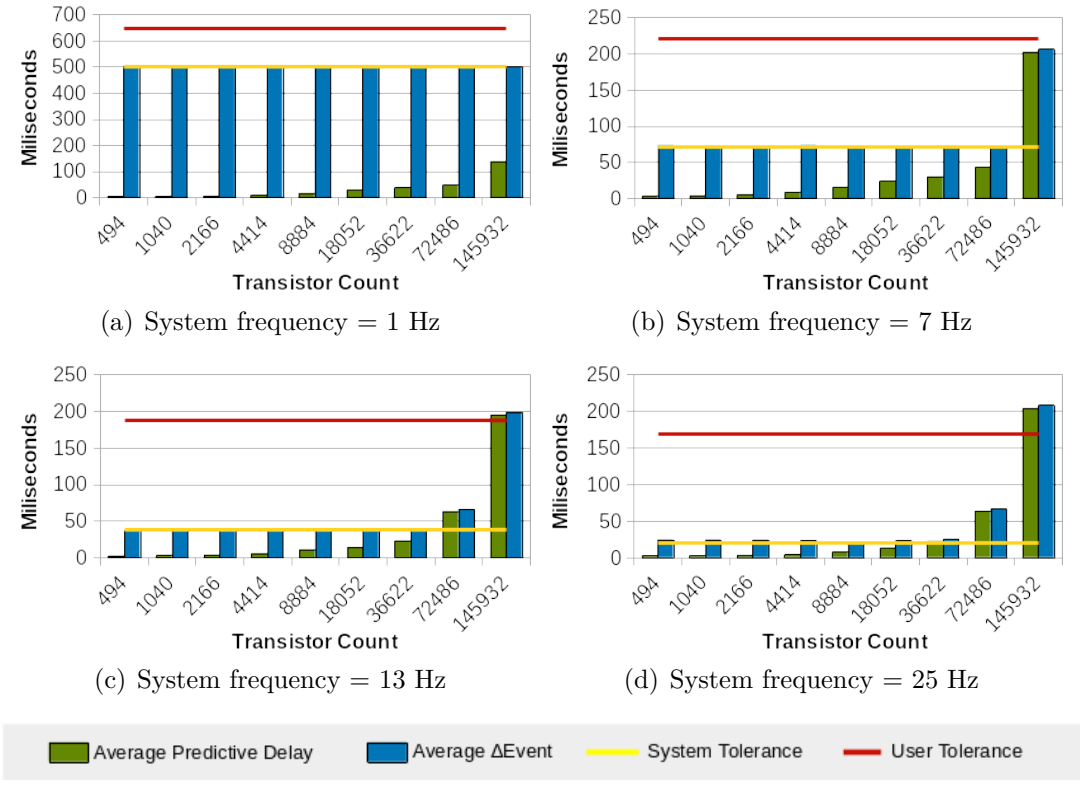
Figure 6.2 shows four charts for different N-bit counters running at 1, 7, 13, and 25 Hz system frequencies. The 494 number on the leftmost side of the chart in

Figure 6.2(a) represents the number of transistors for an 8-bit counter synthesized with the Pitanga platform. The numbers to the right also represent the number of transistors of N-bit counters. Each number is associated with an N-bit counter with twice as bits as the left one. That is, the 1040-transistor counter has 16 bits, the 2166-transistor counter has 32 bits, and so on, up to the rightmost counter with 2048 bits.

The chart shows the average predictive delay for N-bit counters ranging from 8 to 2048 bits. Each result is an average of 102 samples. The predictive delay exceeds the system tolerance yellow line at 7Hz for the counter with 145,932 transistors (i.e., the 2048-bit counters). At 25Hz, the N-bit counters with 36,3622 transistors is on the limit of the system tolerance.

The user tolerance is exceeded for the N-bit counter with 145,932 transistors at 13 Hz and 25Hz. In other words, the user perceives the lack of system responsiveness for the 2048-bit counters at 13Hz and 25Hz on single-machine architectures.

Figure 6.2: Average predictive delay and $\Delta Event$ values for different system frequencies (RTT=0).



(a) System frequency = 1 Hz

(b) System frequency = 7 Hz

(c) System frequency = 13 Hz

(d) System frequency = 25 Hz

6.2.2 RTT≠0

The lack of system responsiveness can be observed for smaller circuits and lower frequencies in client-server architectures. Figure 6.3 shows four charts following the same layout as the charts presented in Section 6.2.1. Note that, compared to the charts in Figure 6.2, the rightmost counter of 2,048 bits (i.e., 145,932 transistors) does not appear. This counter had compilation errors in the server-side hardware and, therefore, it was excluded from the analysis set. The N-bit counters of 72,486 and 36,662 were also removed from Figure 6.3(d) to enlarge the analysis chart area.

The average predictive delay indicates that the user tolerance is exceeded for N-bit counters with more than 512 running at 13 Hz. Therefore, the system responsiveness decays in designs with 36,622 transistors or more at 13 Hz. For designs running at 25 Hz, all counters exceed the system tolerance range, indicating a possible lack of responsiveness if the same system runs additional processes on the same CPU.

Figure 6.3: Average *predictive delay* and $\Delta Event$ values for different system frequencies (RTT≠0).

(a) System frequency = 1Hz

(b) System frequency @ 7Hz

(c) System frequency @ 13Hz

(d) System frequency @ 25Hz

6.3 Tolerance Analysis

Section 6.2 presented the system responsiveness for different circuit sizes emulating different operating frequencies. However, the analysis was performed through an average of 102 predictive delay samples. As these data represent the sample average, they provide a general overview of the system behavior.

Sections 6.3.1 and 6.3.2 aim to present a statistical analysis of the sampled data and are more detailed and conclusive. The analysis was performed only for the traditional client-server architecture (RTT$\neq$0).

6.3.1 Histogram

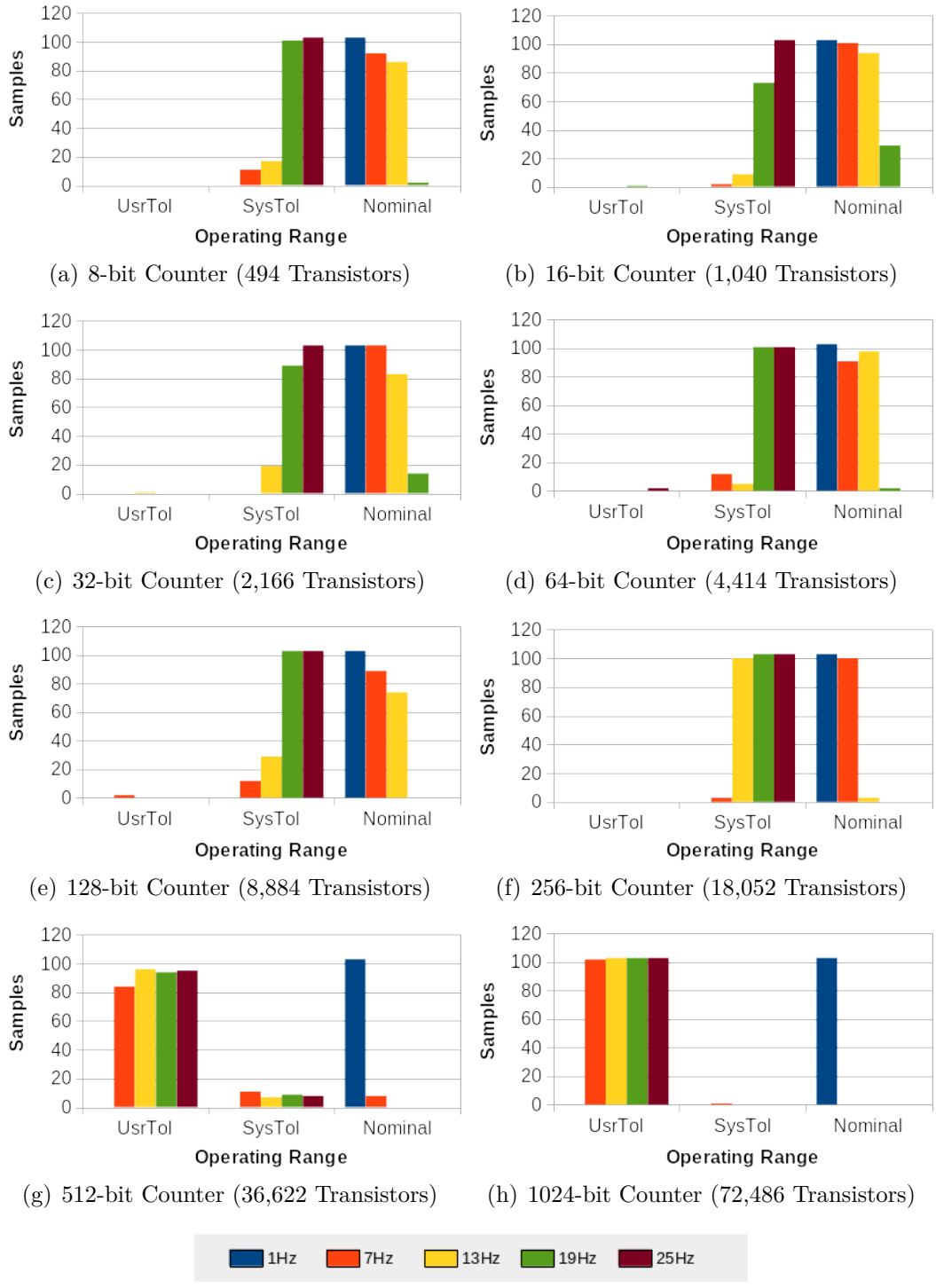
Figure 6.4 shows the operation of the eight N-bit counters in the *Nominal*, *SysTol*, and *UsrTol* operating ranges. Each operation range is indicated on the horizontal axis of the charts, and its meaning is described as follows:

- *Nominal*: indicates that the system is responsive to the user;
- *SysTol*: indicates that the system is at risk of becoming unresponsive to the user if the CPU load increases with the execution of new processes;
- *UsrTol*: indicates that the system is unresponsive to the user.

The analysis of the charts shows that the system is responsive to the user for all the N-bit counters at 1Hz. For counters up to 18,052 transistors (i.e., 256-bit counter or less), the system operates in the SysTol range in some frequencies. In this operation range, the user does not perceive the lack of responsiveness of the system, but the system identifies delays in the arrival of data on the client-side.

Finally, for circuits larger than 36,622 transistors (i.e., 512-bit counters), most of the samples are in the UsrTol range; therefore, the system becomes unresponsive to the user. Some samples can also be found in the UsrTol range in Figures 6.4(b) to 6.4(e). However, due to the small number of samples in this operation range, the system response delay may not be perceptible by the user due to the higher operating frequency.

Figure 6.4: Operating ranges for different counter sizes running in the traditional client-server architeture (RTT≠0).



(a) 8-bit Counter (494 Transistors)

(b) 16-bit Counter (1,040 Transistors)

(c) 32-bit Counter (2,166 Transistors)

(d) 64-bit Counter (4,414 Transistors)

(e) 128-bit Counter (8,884 Transistors)

(f) 256-bit Counter (18,052 Transistors)

(g) 512-bit Counter (36,622 Transistors)

(h) 1024-bit Counter (72,486 Transistors)

6.3.2 Heat map

Figure 6.5 presents the system responsiveness in the heat map format for six operation scenarios of the Pitanga platform. The heat map has different colors to indicate each operating range, including the Nominal, the SysTol, and the UsrTol

operating ranges. The column maps present when the RTT=0 and the RTT≠0. The row maps show results for the Pitanga platform operating in the best-case, nominal-case, and worst-case scenarios. The meaning of each of these cases is described as follows:

- *Best-case*: the maximum slack time of the population of 102 samples;
- *Nominal-case*: the average slack time of the population of 102 samples;
- *Worst-case*: the minimum slack time of the population of 102 samples.

Each column contains slack results for different sizes of N-bit counters according to their number of transistors. The slacks are obtained according to the equations 5.10a and 5.10b in Section 5.4.3.

Figure 6.5: System responsiveness decay for designs with different transistors count for system frequencies ranging from 1 to 25 Hz.

Max - Slack (ms)	System Frequency (Hz)				
Transistor Count	1	7	13	19	25
494	498,32	69,79	36,71	24,57	18,37
1040	497,36	69,65	36,59	24,50	18,04
2166	496,96	68,86	36,02	23,88	17,58
4414	495,47	67,66	34,83	22,74	16,60
8884	493,81	65,58	32,55	20,49	14,15
18052	489,10	60,10	27,65	15,55	9,09
36622	479,21	50,39	17,66	5,75	-0,64
72486	460,21	30,83	0,11	-12,33	-18,11

(a) Best-case scenario (RTT=0)

Max - Slack (ms)	System Frequency (Hz)				
Transistor Count	1	7	13	19	25
494	473,84	47,85	15,73	1,11	-3,76
1040	474,85	47,45	15,98	3,97	-5,03
2166	474,74	46,37	13,65	2,89	-6,13
4414	472,00	42,17	11,38	1,06	-5,70
8884	469,36	40,82	8,00	-2,94	-7,73
18052	462,36	37,98	1,54	-7,02	-15,92
36622	447,86	18,61	-15,56	-26,52	-34,10
72486	420,51	-141,13	-191,76	-217,59	-206,24

(b) Best-case scenario (RTT≠0)

Avg - Slack (ms)	System Frequency (Hz)				
Transistor Count	1	7	13	19	25
494	495,55	67,84	35,73	23,88	17,50
1040	494,93	67,11	35,33	23,71	17,20
2166	492,79	65,39	34,51	22,94	16,26
4414	488,84	62,51	32,32	22,12	15,53
8884	482,46	55,99	27,22	18,67	12,17
18052	471,21	46,78	24,01	14,02	7,36
36622	461,40	40,90	15,03	1,65	-2,86
72486	452,14	28,06	-25,23	-37,39	-44,15

(c) Nominal scenario (RTT=0)

Avg - Slack (ms)	System Frequency (Hz)				
Transistor Count	1	7	13	19	25
494	461,74	29,89	5,38	-6,12	-13,22
1040	464,52	39,53	8,84	-9,58	-10,24
2166	464,51	39,04	-6,28	-2,71	-13,93
4414	459,42	26,74	6,31	-5,12	-21,02
8884	454,13	15,35	-0,94	-10,72	-16,22
18052	448,07	27,87	-5,94	-17,28	-30,20
36622	433,83	-154,30	-251,18	-202,55	-213,73
72486	399,25	-380,85	-413,95	-654,88	-437,60

(d) Nominal scenario (RTT≠0)

Min - Slack (ms)	System Frequency (Hz)				
Transistor Count	1	7	13	19	25
494	493,58	55,22	33,91	22,91	15,39
1040	493,53	63,30	31,78	22,48	14,91
2166	490,89	51,08	29,81	21,36	10,09
4414	487,09	56,53	25,40	19,43	12,80
8884	479,14	50,86	18,55	13,16	6,86
18052	462,12	21,99	13,71	2,27	-0,98
36622	442,80	19,05	-0,60	-14,76	-18,77
72486	431,87	-9,59	-117,37	-118,65	-144,80

(e) Worst-case scenario (RTT=0)

Min - Slack (ms)	System Frequency (Hz)				
Transistor Count	1	7	13	19	25
494	345,32	-110,83	-44,28	-45,81	-113,54
1040	360,22	-13,66	-20,49	-150,97	-24,53
2166	371,23	21,85	-154,15	-17,06	-108,94
4414	335,39	-121,46	-7,63	-14,57	-307,75
8884	360,80	-631,47	-83,54	-37,79	-47,63
18052	337,10	-19,09	-44,17	-65,10	-145,47
36622	314,17	-254,42	-959,77	-301,85	-467,20
72486	4,85	-664,68	-692,66	-1470,67	-711,95

(f) Worst-case scenario (RTT≠0)

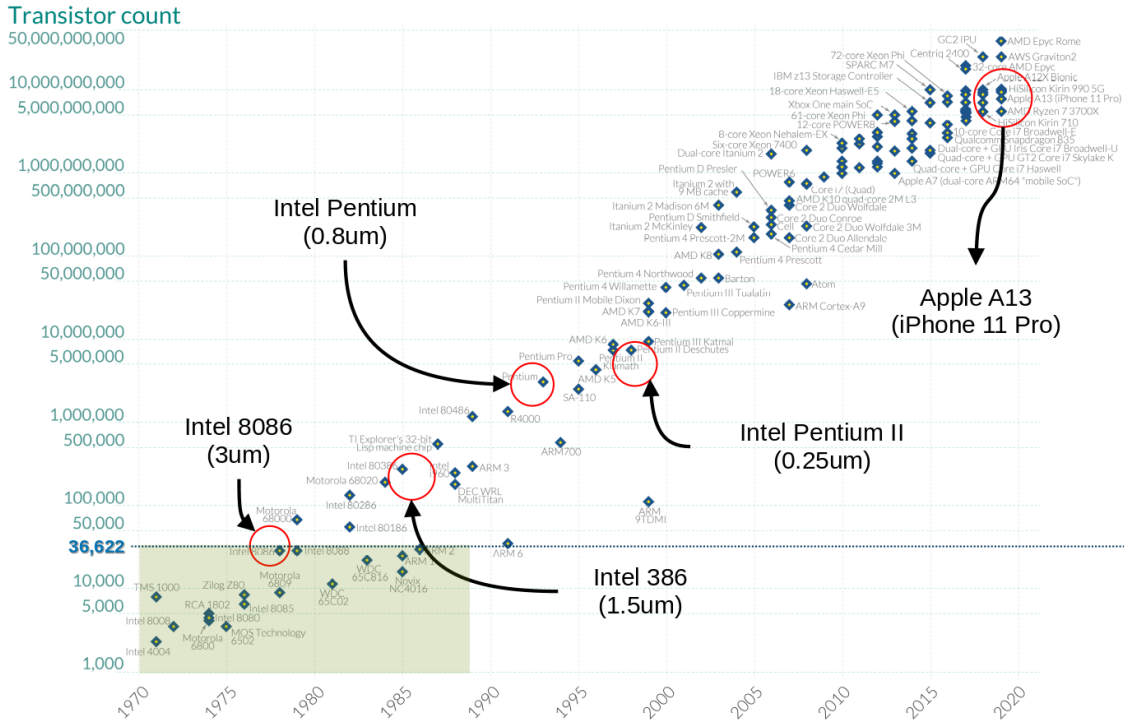
Nominal SysTol UsrTol

The heat map analysis shows that the system responsiveness is inversely proportional to the operating frequency and the number of emulated transistors. Therefore, the analysis confirms the hypothesis of the system responsiveness decay proposed in Figure 1.3.

6.4 Comparison with past commercial IC designs

Figure 6.6 shows the evolution of various commercial ICs designs according to the number of transistors (ROSER; RITCHIE; MATHIEU, 2023). Red circles highlight some ICs to assist the reader in understanding the evolution of CPUs over time. The same chart presents a green rectangle in the lower left corner, limited to 36,622 transistors on the vertical axis. This quantity represents the equivalent number of CMOS transistors obtained by the Pitanga platform for a 1,024-bit counter using the Pitanga library described in Table 5.1.

Figure 6.6: Emulation capacity of the Pitanga platform compared to past IC designs. The highlighted number in the vertical axis represents the number of transistors for a 1,024-bit counter designed in the Pitanga platform using the Pitanga library.

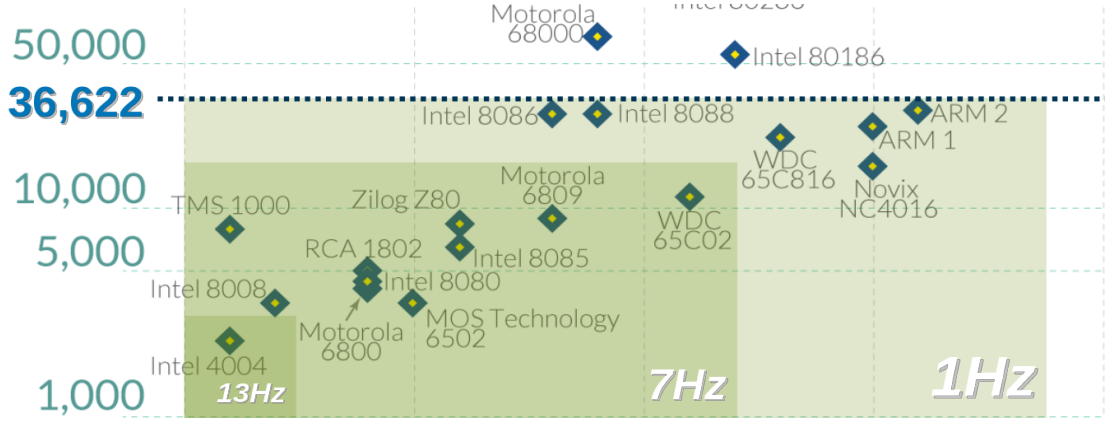


By enlarging the green rectangle highlighted in Figure 6.6, we obtain Figure 6.7, where we observe several historical ICs from the past, including:

- *Intel 4004*: The first CPU released by Intel in 1971 (INTEL4004, 2023) was used by the Busicom 141-PF printing calculators of the Nippon Calculating Machine Corporation. Fabricated using NMOS technology, the Intel 4004 CPU has 2,300 transistors. This number of transistors is equivalent to 5,600 CMOS transistors, a complexity compared to a 64-bit counter on the Pitanga platform.

- *MOS Technology 6502*: With 3,510 transistors (WIKIPEDIA, 2023b), variations of this CPU were used in the Atari 2600 and Nintendo Entertainment System (NES) consoles. These versions employed NMOS technology, thus, half the number of transistors compared to CMOS technologies. The number of transistors in the 6502 can be compared to $2 \cdot 3,510 = 7,020$ CMOS transistors, which is close to the capacity of a 128-bit counter on the Pitanga platform.
- *Zilog Z80*: The Z80A version of this CPU was widely used in Sega Master System consoles (WIKIPEDIA, 2023e). Developed with 8,500 NMOS transistors by Zilog in 1974, this thesis compares it to a 256-bit counter or a near-equivalent containing $2 \cdot 8,500 = 17,000$ CMOS transistors on the Pitanga platform.
- *WDC 65C816*: The 22,000 transistors implemented in CMOS technology (WIKIPEDIA, 2023d) is an improved version of the MOS Technology 6502 CPU and had several variations, including the Ricoh 5A22 CPU (WIKIPEDIA, 2023c) used in the Super Nintendo Entertainment System (SNES). A 512-bit counter emulated on the Pitanga platform has more CMOS transistors than this CPU.
- *Intel 8086*: Known for its instruction set, which persists in the CPUs of most contemporary PCs, the Intel 8086 was released in 1978 with 29,000 NMOS transistors (WIKIPEDIA, 2023a). A 1024-bit counter on the Pitanga platform can emulate 72,486 CMOS transistors.

Figure 6.7: Emulation capacity of the Pitanga platform compared to real designs.



Considering the Pitanga platform operating in the nominal scenario defined in Figure 6.4(d)), the Intel 4004, MOS Technology 6502, Zilog Z80, WDC 65C816,

and Intel 8086 CPUs can be emulate at 13Hz, 7Hz, 7Hz, 1Hz, and 1Hz frequencies, respectively. In other words, not only can the CPUs of the Atari 2600, NES, Master System, and SNES be used for learning hardware development on the Pitanga platform, but they can also be emulated.

6.5 Contributions of this chapter

This chapter presented the results of the Pitanga emulation-based remote laboratory. Following to the methodology defined in Chapter 5, this chapter showed that the predictive emulator runtime is $O(n)$. Additionally, through 102 collected samples, we identified three operating ranges for the platform: Nominal, SysTol, and UsrTol. Based on these operating ranges, we classified a group of designs according to their operating frequency and transistor count on a heat map. The heat map confirmed the hypothesis of the system responsiveness decay proposed in Figure 1.3 and allowed the comparison of the results with historical CPU designs from the past. Finally, the chapter concludes that the Pitanga platform can emulate various CPUs used in the 80s video game consoles, provided they run at frequencies around 10Hz.

7 CONCLUSIONS AND FUTURE WORKS

This thesis claims that students can experiment with responsive digital circuits remotely, without FPGA-based prototyping boards, using industry standards in a cost-efficient, easy-to-use design environment. In order to support this thesis, Chapter 2 contextualizes the reader by explaining the foundational concepts of VLSI design methodologies and flows commonly used in the industry. Chapter 3 presents the evolution of digital circuit remote laboratories on education, identifying the solutions and limitations of FPGA-based and simulation-based remote laboratories. Chapter 4 delves deep into the limitations of state-of-art FPGA-based and simulation-based remote laboratories and introduces a third option not explored in education: emulation-based remote laboratories. Still, in Chapter 4, an emulation-based architecture using general-purpose CPUs and industry standards is proposed as an alternative for FPGA-based laboratories, providing a cost-efficient laboratory capable of conducting responsive experiments over the Internet.

The thesis presents the emulation-based remote laboratory architecture, the Pitanga platform, that implements a particular communication interface and an AIG-based emulator on the server-side. This implementation enables a responsive platform over the Internet by predicting the output values for every possible input state from the client-side. The algorithm capable of accomplishing this feature is named predictive emulator, and Chapters 5 and 6 present the methodology and results for this algorithm, respectively.

7.1 Contributions of this thesis

The thesis proposes a list of requirements for responsive digital circuits remote laboratories as defined in Table 1.1. The proposed remote laboratory must be: based on industry standards, inexpensive, and easy-to-use. This thesis has made contributions to these criteria, as indicated below:

- *based on industry standards*: The Pitanga platform supports the Verilog standard for digital circuit description, follows a design flow that resembles professional tools using a cell library during RTL synthesis, and displays design summary reports similar to professional tools. The Verilog standard allows the

platform to integrate with industry software, such as the Yosys RTL synthesis tools (requirements 1, 2, and 3).

- *inexpensive*: The platform runs on general-purpose CPUs, decreasing the need for educational institutions and students to acquire or replace legacy educational FPGA-based prototyping boards. Moreover, the platform supports scaling on the server-side according to user demand, allowing additional cost-affordable CPU power as needed. As the emulation platform is a software-based model, the risks of hardware failure decrease significantly, reducing also the costs of maintaining a permanent laboratory maintenance staff (requirements 4, 5, and 6).
- *easy-to-use*: The platform provides a virtual prototyping board on the client-side that closely resembles the PCB layouts of entry-level educational FPGA-based prototyping boards. The client-side size is 133MB after installation, much smaller than the gigabytes of FPGA design software for digital circuit design and implementation. With the development of documentation and teaching materials, various educational institutions can benefit from the platform because it can be easily installed without needing additional physical hardware (requirements 7, 8, and 9).

Satisfying requirements 1 to 9 are commercial contributions of this thesis. However, from a scientific point of view, the main contribution is the solution to the tenth requirement in table 1.1: a method for providing real-time user stimuli response.

The real-time response is one of the limitations imposed by FPGA-based remote laboratories. The state-of-art solution for this issue is a pre-recorded database of past experiments on video. However, this solution is possible only for experiments with limited inputs and conditions. This thesis in Chapter 2 explored EDA modern data structures, circuit simulation, and multiplayer game programming techniques to broaden this limitation. As a result, the thesis delivers a software solution capable of predicting all the possible outputs for a given input state of a digital circuit description in Verilog. The solution uses a predictive emulator algorithm that computes all the possible outputs using an AIG-based data structure in $O(n)$ time.

Further analysis of the predictive emulator shows that the size and operating frequency of the input circuit determine the operating range in which the circuit may run without noticeable latency. Latency occurs when the predictive emulator cannot

deliver the results before the subsequent user event or clock event. Thus, as the final contribution of Chapter 6, this thesis produces a heat map categorizing operating ranges for different circuit sizes and frequencies running in the Pitanga platform. The heat map confirms the hypothesis of the expected system responsiveness decay depicted in Figure 1.3.

7.2 Future works

Every new solution has its limitations, and the solution proposed in this thesis is no different. The proposed emulation-based remote laboratory has limitations. Some limitations are straightforward to solve, while others are not. Some limitations can be more complex to solve. However, every limitation may become an opportunity for future improvements.

One of the limitations that can be easily solved is the low operating frequency. The results obtained in this thesis indicate that the main contribution, which is the predictive emulator, operates near 10 Hertz. This frequency is enough for students development basic sequential digital circuits, but it limits more advanced students who seek to develop more complex circuits. One way to quickly increase the operating frequency of the emulator to the hundreds of Hertz range is to rewrite the Pitanga platform code to a compiled language. The results obtained for this thesis run on interpreted Python without any code acceleration treatment. Another way to accelerate the predictive emulator is to use machines with better computational performance on the server-side. Consider that the results in Chapter 6 ran on machines with low computational performance.

More complex solutions to increase emulation speed, potentially reaching thousands of Hertz, could involve combining the previous solutions with an interrupt system on the server-side. Applications that operate at higher frequencies do not require an immediate response and can wait for the server to complete a given operation. This implementation would allow the server to execute several clock cycles at each event occurs on the client side, differentiating the clock speed of the client and the server sides. This solution requires the implementation of a synchronization scheme that could be adapted from those used in multiplayer games. Another solution is running the client and server on the same machine. In this case, the situation returns to leaving the user responsible for installing and configuring

the design software environment, making it appropriate for more advanced users.

Examples of more challenging solutions that could improve emulator performance include identifying and separating digital circuit blocks into different CPU cores. Developing this solution requires a well-implemented inter-process communication system and an algorithm for partitioning the AIG into different CPU cores. Other solutions involve developing compilers that identify arithmetic constructs that do not need to be executed with the help of an AIG. For instance, instructions such as addition could replace a large portion of an AIG that performs an addition operation with a single instruction in the target CPU architecture.

Finally, there are also impractical implementation strategies, such as increasing the hamming distance of the predictive emulator too much. In this case, the emulator would predict sequences of possible user interactions, such as pressing a sequence of buttons rather than just one. In such cases, the calculation time of the predictive emulator becomes exponential, making this solution impractical. However, depending on certain conditions, such as when the inputs to the virtual prototyping board are few, increasing the depth of the Hamming distance of the predictive emulator may become feasible.

8 APPENDIX

Table 8.1: References for the Table 3.1 listed in Chapter 3

Software	Reference
CircuitVerse	(IIIT-Bangalore, 2022)
logic.ly	(logic.ly, 2022)
LogiSim	(Carl Burch, 2022)
Deeds	(Giuliano Donzellini, 2022)
Icarus	(Stephen Williams, 2022)
Verilator	(CHIPS Alliance, 2022)
LogicCircuit	(Logic Circuit, 2022)
simulator IO	(Bastian Born, 2022)
Logic Gate	(Steven Kollmansberger, 2022)
OpenCircuit	(OpenCircuits, 2022)
SmartSim	(Ashley Newson, 2022)
BOOLR	(Jaap Dechering, Gees Brouwer, Teun de Theije, 2022)
LogiJS	(LogiJS, 2022)
EasySim	(EasySim, 2022)
wiRedPanda	(GIBIS UNIFESP, 2022)
Digital	(Helmut Neemann, 2022)
Hradla	(Jenda Horák, 2022)
MAX+Plus II	(INTEL, 2022)
Intel Quartus	(INTEL, 2022)
Xilinx Vivado	(AMD Xilinx, 2022)
Logic Circuit Pro	(Stefan Belinov, 2022)
EDA playground	(DOULOS, 2022)
#Data	(Von Braun Labs, 2022)

REFERENCES

ACCELLERA. **Available IEC/IEEE Standards**. 2022. <https://accellera.org/ieee-1850/ieee-1850-issues/hm/0049.html>. Accessed: 2022-12-01.

AHO, A. et al. **Compilers: Principles, Techniques, and Tools**. Addison-Wesley, 2007. (Alternative eText Formats Series). ISBN 9780321547989. Available from Internet: <https://books.google.com.br/books?id=WomBPgAACAAJ>.

AKTAN, B. et al. Distance learning applied to control engineering laboratories. **IEEE Transactions on Education**, v. 39, n. 3, p. 320–326, 1996. Available from Internet: <http://ieeexplore.ieee.org/document/538754/>.

Al Qassem, L. M. et al. A remote FPGA laboratory as a cloud microservice. **Proceedings - IEEE International Symposium on Circuits and Systems**, Institute of Electrical and Electronics Engineers Inc., v. 2020-Octob, 2020. ISSN 02714310. Available from Internet: <https://ieeexplore.ieee.org/abstract/document/9181123>.

ALDEC. **TechnologyUVM Simulation Acceleration**. 2022. <https://www.aldec.com/en/solutions/hardware_emulation_solutions/acceleration>. Accessed: 2022-12-04.

ALMEIDA, F. V. de et al. Teaching digital electronics during the covid-19 pandemic via a remote lab. **Sensors 2022, Vol. 22, Page 6944**, Multidisciplinary Digital Publishing Institute, v. 22, p. 6944, 9 2022. ISSN 1424-8220. Available from Internet: <https://www.mdpi.com/1424-8220/22/18/6944/htmhttps://www.mdpi.com/1424-8220/22/18/6944>.

AMD Xilinx. **Xilinx - Adaptable. Intelligent.** 2022. Available from Internet: <https://www.xilinx.com/>.

ANGULO, I. et al. Integral Remote laboratory for Programmable Logic. In: 2019 5TH EXPERIMENT INTERNATIONAL CONFERENCE (EXP.AT'19). **Proceedings...** IEEE, 2019. p. 253–255. Available from Internet: <https://ieeexplore.ieee.org/document/8876561/>.

ANGULO, I.; RODRIGUEZ-GIL, L.; GARCIA-ZUBIA, J. Scaling up the Lab: An Adaptable and Scalable Architecture for Embedded Systems Remote Labs. **IEEE Access**, Institute of Electrical and Electronics Engineers Inc., v. 6, p. 16887–16900, mar 2018. Available from Internet: <https://ieeexplore.ieee.org/document/8307384http://ieeexplore.ieee.org/document/8307384/>.

APPLE. **Apple unveils M1 Ultra, the world's most powerful chip for a personal computer**. 2022. <https://www.apple.com/newsroom/2022/03/apple-unveils-m1-ultra-the-worlds-most-powerful-chip-for-a-personal-computer/>.

ASGARI, S. et al. An observational study of engineering online education during the covid-19 pandemic. **PLOS ONE**, Public Library of Science, v. 16, p. e0250041, 4 2021. ISSN 1932-6203. Available from Internet: <https://dx.plos.org/10.1371/journal.pone.0250041>.

Ashley Newson. **SmartSim**. 2022. Available from Internet: <https://smartsim.org.uk/home/>.

BACHRACH, J. et al. Cyclist: Accelerating hardware development. **IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD**, Institute of Electrical and Electronics Engineers Inc., v. 2017-November, p. 1011–1018, 12 2017.

BALAMURALITHARA, B.; WOODS, P. C. Virtual laboratories in engineering education: The simulation lab and remote lab. **Computer Applications in Engineering Education**, John Wiley & Sons, Ltd, 2009.

Bastian Born. **simulator IO**. 2022. Available from Internet: <https://simulator.io/>.

BECKER, J. et al. Internet-based training of reconfigurable technologies. **Proceedings - 11th Brazilian Symposium on Integrated Circuit Design, SBCCI 1998**, Institute of Electrical and Electronics Engineers Inc., v. 1998-Septe, p. 25–30, 1998. Available from Internet: <https://ieeexplore.ieee.org/abstract/document/715404>.

BECKER, J. et al. Providing Flexible Internet Infrastructure for FPGA-Based CAD Courses. **Microelectronics Education**, Springer, Dordrecht, p. 277–280, 2000. Available from Internet: <https://link.springer.com/chapter/10.1007/978-94-015-9506-3_64>.

BIANCOLIN, D. et al. Fased: Fpga-accelerated simulation and evaluation of dram. In: PROCEEDINGS OF THE 2019 ACM/SIGDA INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE GATE ARRAYS. **Proceedings...** [S.l.]: Association for Computing Machinery, 2019. p. 330–339.

BOHUS, C. et al. Running Control Engineering Experiments Over the Internet. **IFAC Proceedings Volumes**, v. 29, n. 1, p. 2919–2927, jun 1996. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1474667017581215https://linkinghub.elsevier.com/retrieve/pii/S1474667017581215>.

BOULDIN, D. Impacting education using fpgas. In: . IEEE, 2004. v. 18, p. 142–147. ISBN 0-7695-2132-0. Available from Internet: <http://ieeexplore.ieee.org/document/1303120/>.

BRANDãO, E. D. et al. Possible reductions to generate circuits from bdds. In: 2022 IEEE COMPUTER SOCIETY ANNUAL SYMPOSIUM ON VLSI (ISVLSI). **Proceedings...** [S.l.: s.n.], 2022. p. 406–409.

BRAYTON, R.; MISHCHENKO, A. Abc: An academic industrial-strength verification tool. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, Springer, Berlin, Heidelberg, v. 6174 LNCS, p. 24–40, 2010. ISSN 03029743. Available from Internet: <https://link.springer.com/chapter/10.1007/978-3-642-14295-6_5>.

BUTZEN, P. et al. Design of cmos logic gates with enhanced robustness against aging degradation. **Microelectronics Reliability**, v. 52, n. 9, p. 1822–1826, 2012. ISSN 0026-2714. SPECIAL ISSUE 23rd EUROPEAN SYMPOSIUM ON THE RELIABILITY OF ELECTRON DEVICES, FAILURE PHYSICS AND ANALYSIS. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0026271412002892>.

BUTZEN, P. F. et al. Transistor network restructuring against nbti degradation. **Microelectronics Reliability**, v. 50, n. 9, p. 1298–1303, 2010. ISSN 0026-2714. 21st European Symposium on the Reliability of Electron Devices, Failure Physics and Analysis. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0026271410004130>.

BUTZEN, P. F. et al. Standby power consumption estimation by interacting leakage current mechanisms in nanoscaled cmos digital circuits. **Microelectronics Journal**, v. 41, n. 4, p. 247–255, 2010. ISSN 0026-2692. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S002626921000042X>.

CADENCE. **Advanced Verification with Specman® Elite, Version 2.0**. 2006. CI Brasil Program.

CADENCE. **Encounter® Conformal® Constraint Designer, Version 7.1**. 2007. CI Brasil Program.

CADENCE. **Logic Equivalence Checking with Encounter® Conformal® EC, Version 7.1**. 2007. CI Brasil Program.

CADENCE. **Specman® Elite Basics for Verification Environment Users, Version 6.1.1**. 2007. CI Brasil Program.

CADENCE. **BD02: Digital IC Design, Version 1.1, Student Handout**. 2008. CI Brasil Program.

CADENCE. **BD03: Digital Physical Design, Version 1.1, Student Handout**. 2008. CI Brasil Program.

CADENCE. **BG01: Semiconductor Business Processes, Version 1.1, Student Handout**. 2008. CI Brasil Program.

CADENCE. **Encounter® RTL Compiler, Version 7.2**. 2008. CI Brasil Program.

CADENCE. **Signoff Timing Analysis with Encounter® Timing System, Version 7.1**. 2008. CI Brasil Program.

CADENCE. **VoltageStorm Power Rail Analysis, Version 7.1**. 2008. CI Brasil Program.

CADENCE. **VoltageStormAssura Verification, Version 3.1.4**. 2008. CI Brasil Program.

CADENCE. **Cadence Expands Collaboration with TSMC and Microsoft to Accelerate Timing Signoff for Giga-Scale Designs on the Cloud**. 2021. <https://www.cadence.com/en_US/home/company/newsroom/press-releases/pr/2021/cadence-expands-collaboration-with-tsmc-and-microsoft-to-acceler.html>. Accessed: 2022-11-26.

CADENCE. **AMD Designs 3rd-Gen EPYC Server Processors for HPC with Dynamic Duo**. 2022. Available from Internet: <https://www.cadence.com/en_US/home/multimedia.html/content/dam/cadence-www/global/en_US/videos/solutions/amd-designed-with-cadence.mp4>.

CADENCE. **NVIDIA Embraces the Groundbreaking Technology, Dynamic Duo 2.0**. 2022. Available from Internet: <https://www.cadence.com/en_US/home/multimedia.html/content/dam/cadence-www/global/en_US/videos/solutions/nvidia-designed-with-cadence-dynamic-duo.mp4>.

CADENCE. **Palladium Emulation**. 2022. <https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulation-and-prototyping/palladium.html>. Accessed: 2022-12-05.

CADENCE. **Palladium Hybrid**. 2022. <https://www.cadence.com/ko_KR/home/tools/system-design-and-verification/acceleration-and-emulation/palladium-hybrid.html>. Accessed: 2022-12-04.

CAI, M.; LUO, J. Influence of covid-19 on manufacturing industry and corresponding countermeasures from supply chain perspective. **Journal of Shanghai Jiaotong University (Science) 2020 25:4**, Springer, v. 25, p. 409–416, 8 2020. ISSN 1995-8188. Available from Internet: <https://link.springer.com/article/10.1007/s12204-020-2206-z>.

Carl Burch. **Logisim**. 2022. Available from Internet: <http://www.cburch.com/logisim/>.

CHAKRAVARTHI, V. **A Practical Approach to VLSI System on Chip (SoC) Design: A Comprehensive Guide**. [S.l.]: Springer International Publishing, 2019.

CHINNERY, D. et al. Design flows. In: _____. ELECTRONIC DESIGN AUTOMATION FOR IC IMPLEMENTATION, CIRCUIT DESIGN, AND PROCESS TECHNOLOGY. **Proceedings...** [S.l.]: CRC Press, 2017. p. 3–25.

CHIPS Alliance. **Verilator**. 2022. Available from Internet: <https://www.veripool.org/verilator/>.

CIOTTI, M. et al. The covid-19 pandemic. **Critical Reviews in Clinical Laboratory Sciences**, Taylor and Francis Ltd., p. 365–388, 2020. ISSN 1549781X.

Claire Xenia Wolf. **yosys – Yosys Open SYnthesis Suite**. 2023. Available from Internet: <https://github.com/YosysHQ/yosys>.

CLOUD, G. **Compute Engine general-purpose machine family**. 2023. Accessed: 2023-04-16.

CORREIA, V.; REIS, A. Advanced technology mapping for standard-cell generators. In: PROCEEDINGS OF THE 17TH SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN. **Proceedings...** New York, NY, USA: Association for Computing Machinery, 2004. (SBCCI '04), p. 254–259. ISBN 1581139470. Available from Internet: <https://doi.org/10.1145/1016568.1016636>.

COSTA, A.; DROVES, L.; REIS, A. **Sistemas para emulação de hardware**. 2023. Applicant: inPlace Softwares para Automação LTDA and Universidade Federal do Rio Grande do Sul. BR 10 2023 008512 1. Application date: May 3rd, 2023. Patent pending.

COSTA, A.; DROVES, L.; REIS, A. **Sistemas para emulação de hardware com anúncios associados**. 2023. Applicant: inPlace Softwares para Automação LTDA and Universidade Federal do Rio Grande do Sul. BR 10 2023 008529 6. Application date: May 3rd, 2023. Patent pending.

COSTA, A.; DROVES, L.; REIS, A. **Sistemas para emulação de hardware como serviço de software**. 2023. Applicant: inPlace Softwares para Automação LTDA and Universidade Federal do Rio Grande do Sul. BR 10 2023 008528 8. Application date: May 3rd, 2023. Patent pending.

COSTA, A.; DROVES, L.; REIS, A. **Sistemas para emulação preditiva de hardware**. 2023. Applicant: inPlace Softwares para Automação LTDA and Universidade Federal do Rio Grande do Sul. BR 10 2023 008527 0. Application date: May 3rd, 2023. Patent pending.

COSTA, A. S.; SILVEIRA, L. D.; REIS, A. I. A virtual board approach for prototyping and teaching digital design. In: 2022 35TH SBC/SBMICRO/IEEE/ACM SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI). **Proceedings...** [S.l.: s.n.], 2022. p. 1–6.

COSTA, A. S.; SILVEIRA, L. D.; REIS, A. I. Live demonstration: Pitanga platform for virtual fpga remote laboratories. In: 2023 IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS). **Proceedings...** [S.l.: s.n.], 2023. p. 1–5.

da Silva, D. N.; REIS, A. I.; RIBAS, R. P. Cmos logic gate performance variability related to transistor network arrangements. **Microelectronics Reliability**, v. 49, n. 9, p. 977–981, 2009. ISSN 0026-2714. 20th European Symposium on the Reliability of Electron Devices, Failure Physics and Analysis. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0026271409002546>.

DILLINGER, T. **VLSI Design Methodology Development**. [S.l.]: Pearson Education, 2019.

DOSHI, J. et al. Implementing a cloud based Xilinx ISE FPGA design platform for integrated remote labs. **2015 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2015**, Institute of Electrical and Electronics Engineers Inc., p. 533–537, sep 2015. Available from Internet: <https://ieeexplore.ieee.org/abstract/document/7275663>.

DOULOS. **EDA playground**. 2022. Available from Internet: <https://www.edaplayground.com/>.

EasySim. **EasySim**. 2022. Available from Internet: <http://www.research-systems.com/easysim/easysim.htm>.

EDN. **Gate level simulations: verification flow and challenges**. 2014. <https://www.edn.com/gate-level-simulations-verification-flow-and-challenges/>.

EL-MEDANY, W. M. FPGA remote laboratory for hardware E-learning courses. **Proceedings - 2008 IEEE Region 8 International Conference on Computational Technologies in Electrical and Electronics Engineering, SIBIRCON 2008**, p. 106–109, 2008. Available from Internet: <https://ieeexplore.ieee.org/document/4602596>.

FJELDLY, T. A.; SHUR, M. S. **Lab on the Web**. John Wiley & Sons, Inc., 2003. 1–239 p. ISBN 9780471727705. Available from Internet: <https://onlinelibrary.wiley.com/doi/book/10.1002/0471727709http://doi.wiley.com/10.1002/0471727709>.

FJELDLY, T. A.; SHUR, M. S. Lab on the web: Running real electronics experiments via the internet. **Lab on the Web: Running Real Electronics Experiments via the Internet**, Wiley-IEEE Press, p. 1–239, 1 2005. Available from Internet: <https://onlinelibrary.wiley.com/doi/book/10.1002/0471727709>.

FROYD, J. E.; WANKAT, P. C.; SMITH, K. A. Five major shifts in 100 years of engineering education. **Proceedings of the IEEE**, v. 100, n. SPL CONTENT, p. 1344–1360, 2012. Available from Internet: <https://ieeexplore.ieee.org/document/6185632>.

GANGADHARAN, S.; CHURIWALA, S. **Constraining Designs for Synthesis and Timing Analysis: A Practical Guide to Synopsys Design Constraints (SDC)**. [S.l.]: Springer Science & Business Media, 2014.

GAUDIOT, J.-L.; KASAHARA, H. Computer education in the age of covid-19. **Computer**, IEEE Computer Society, v. 53, p. 114–118, 10 2020. ISSN 0018-9162. Available from Internet: <https://ieeexplore.ieee.org/document/9206411/>.

GEREZ, S. **Algorithms for VLSI Design Automation**. 1. ed. [S.l.]: John Wiley & Sons Ltd, 1999.

GIBIS UNIFESP. **wiRedPanda**. 2022. Available from Internet: <https://github.com/GIBIS-UNIFESP/wiRedPanda>.

Giuliano Donzellini. **Digital Electronics Deeds**. 2022. Available from Internet: <https://www.digitalelectronicsdeeds.com/>.

GLAZER, J.; MADHAV, S. **Multiplayer game programming: Architecting networked games**. [S.l.]: Addison-Wesley Professional, 2015.

GOMES, I. A. C. et al. Methodology for achieving best trade-off of area and fault masking coverage in atmr. In: 2014 15TH LATIN AMERICAN TEST WORKSHOP - LATW. **Proceedings...** [S.l.: s.n.], 2014. p. 1–6.

GOMES, I. A. C. et al. Using only redundant modules with approximate logic to reduce drastically area overhead in tmr. In: 2015 16TH LATIN-AMERICAN TEST SYMPOSIUM (LATS). **Proceedings...** [S.l.: s.n.], 2015. p. 1–6.

GOMES, L.; BOGOSYAN, S. Current Trends in Remote Laboratories. **IEEE Transactions on Industrial Electronics**, v. 56, n. 12, p. 4744–4756, dec 2009. Available from Internet: <http://ieeexplore.ieee.org/document/5280206/>.

GOOGLE. **Books Ngram Viewer**. 2022. <https://books.google.com/ngrams/>. Accessed: 2022-11-24.

GUO, M.; HUSSEIN, R.; ORDUNA, P. Rhl-butterfly: A scalable iot-based breadboard prototype for embedded systems laboratories. In: . [S.l.: s.n.], 2022. v. 2022-October.

HASHEMIAN, R.; RIDDLEY, J. FPGA e-Lab, a technique to remote access a laboratory to design and test. **Proceedings - MSE 2007: 2007 IEEE International Conference on Microelectronic Systems Education: Educating Systems Designers for the Global Economy and a Secure World**, p. 139–140, 2007. Available from Internet: <https://ieeexplore.ieee.org/document/4231487>.

Helmut Neemann. **Digital**. 2022. Available from Internet: <https://github.com/hneemann/Digital>.

HUTTON, M.; BETZ, V.; ANDERSON, J. Fpga synthesis and physical design. In: _____. ELECTRONIC DESIGN AUTOMATION FOR IC IMPLEMENTATION, CIRCUIT DESIGN, AND PROCESS TECHNOLOGY. **Proceedings...** [S.l.]: CRC Press, 2016. p. 373–413.

IEEE. Ieee standard for systemverilog–unified hardware design, specification, and verification language. **IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)**, p. 1–1315, 2018.

IEEE. Ieee standard for design and verification of low-power, energy-aware electronic systems. **IEEE Std 1801-2018**, p. 1–548, 2019.

IIIT-Bangalore. **CircuitVerse**. 2022. Available from Internet: <https://circuitverse.org/>.

INDRUSIAK, L. S.; GLESNER, M.; REIS, R. On the Evolution of Remote Laboratories for Prototyping Digital Electronic Systems. **IEEE Transactions on Industrial Electronics**, v. 54, n. 6, p. 3069–3077, dec 2007. Available from Internet: <http://ieeexplore.ieee.org/document/4376288/>.

inPlace Design Automation. **Design digital circuits with NO FPGA**. 2023. Available from Internet: <https://en.inplace-da.com/>.

INTEL. **Intel® Quartus® Prime Software**. 2022. Available from Internet: <https://www.intel.com/>.

INTEL4004. **Intel's First Microprocessor**. 2023. <https://www.intel.com/content/www/us/en/history/museum-story-of-intel-4004.html>.

IZUMI, H. et al. Proposal of the web-based training system for the experiment of the digital circuit. **IECON Proceedings (Industrial Electronics Conference)**, v. 1, p. 1766–1770, 2001.

Jaap Dechering, Gees Brouwer, Teun de Theije. **BOOLR**. 2022. Available from Internet: <http://boolr.me/>.

JAFFE, S. Work from home during and after covid-19. In: . IEEE, 2021. p. 28–28. ISBN 978-1-6654-4476-7. Available from Internet: <https://ieeexplore.ieee.org/document/9474823/>.

Jenda Horák. **Hradla**. 2022. Available from Internet: <https://github.com/janjaromirhorak/hradla>.

JUNIOR, L. S. da R. et al. Fast disjoint transistor networks from bdds. In: PROCEEDINGS OF THE 19TH ANNUAL SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN. **Proceedings...** New York, NY, USA: Association for Computing Machinery, 2006. (SBCCI '06), p. 137–142. ISBN 1595934790. Available from Internet: <https://doi.org/10.1145/1150343.1150381>.

KANG, S.-M. S.; LEBLEBIGI, Y. **CMOS Digital Integrated Circuits: Analisys and Design**. [S.l.]: McGraw-Hill, 2003.

Kivy. **Kivy: The Open Source Python App development Framework.** 2023. Available from Internet: <https://kivy.org/>.

LABSLAND. **LabsLand**. 2022. Available from Internet: <https://labsland.com/>.

LIEBOWITZ, J. Life as a professor amid covid-19. **Computer**, IEEE Computer Society, v. 53, p. 126–128, 12 2020. ISSN 0018-9162. Available from Internet: <https://ieeexplore.ieee.org/document/9269890/>.

Logic Circuit. **Logic Circuit**. 2022. Available from Internet: <https://www.logiccircuit.org/>.

logic.ly. **logic.ly**. 2022. Available from Internet: <https://logic.ly/>.

logiCS. **Logic Circuit Synthesis Lab**. 2023. Available from Internet: <https://www.inf.ufrgs.br/logics/>.

LogiJS. **LogiJS**. 2022. Available from Internet: <https://logijs.com/>.

MACHADO, L. et al. Kl-cut based digital circuit remapping. In: NORCHIP 2012. **Proceedings...** [S.l.: s.n.], 2012. p. 1–4.

MACMILLEN, D. et al. An industrial view of electronic design automation. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 19, p. 1428–1448, 2000. ISSN 02780070.

MADHAV, S. **Game Programming Algorithms and Techniques: A Platform-agnostic Approach**. [S.l.]: Addison Wesley, 2014.

MAGINOT, S. Evaluation criteria of hdls: Vhdl compared to verilog, udl/i & m. **European Design Automation Conference**, Publ by IEEE, p. 746–751, 1992.

MARRANGHELLO, F. S. et al. Factored forms for memristive material implication stateful logic. **IEEE Journal on Emerging and Selected Topics in Circuits and Systems**, v. 5, n. 2, p. 267–278, 2015.

MARTINS, M. et al. Open cell library in 15nm freepdk technology. In: PROCEEDINGS OF THE 2015 SYMPOSIUM ON INTERNATIONAL SYMPOSIUM ON PHYSICAL DESIGN. **Proceedings...** New York, NY, USA: Association for Computing Machinery, 2015. (ISPD '15), p. 171–178. ISBN 9781450333993. Available from Internet: <https://doi.org/10.1145/2717764.2717783>.

MARTINS, M. G. A.; RIBAS, R. P.; REIS, A. I. Functional composition: A new paradigm for performing logic synthesis. In: THIRTEENTH INTERNATIONAL SYMPOSIUM ON QUALITY ELECTRONIC DESIGN (ISQED). **Proceedings...** [S.l.: s.n.], 2012. p. 236–242.

MAYOZ, C. A. et al. FPGA remote laboratory: experience of a shared laboratory between UPNA and UNIFESP. In: 2020 XIV TECHNOLOGIES APPLIED TO ELECTRONICS TEACHING CONFERENCE (TAEE). **Proceedings...** IEEE, 2020. p. 1–8. Available from Internet: <https://ieeexplore.ieee.org/document/9163773https://ieeexplore.ieee.org/document/9163773/>.

MCCRACKEN, S.; ZILIC, Z.; CHAN, H. Real Laboratories for Distance Education. **Journal of Computing and Information Technology**, University of Zagreb, Faculty of Electrical Engineering and Computing, v. 11, n. 1, p. 67, mar 2003. Available from Internet: <https://hrcak.srce.hr/en/clanak/69392http://cit.srce.unizg.hr/index.php/CIT/article/view/1498>.

MELOSIK, M. et al. Remote prototyping of fpga-based devices in the iot concept during the covid-19 pandemic. **Electronics**, Multidisciplinary Digital Publishing Institute, v. 11, p. 1497, 5 2022. ISSN 2079-9292. Available from Internet: <https://www.mdpi.com/2079-9292/11/9/1497>.

MOORE, G. Cramming More Components onto Integrated Circuits. **Electronics**, 1965.

MOREIRA, M. et al. Semi-custom ncl design with commercial eda frameworks: Is it possible? In: 2014 20TH IEEE INTERNATIONAL SYMPOSIUM ON ASYNCHRONOUS CIRCUITS AND SYSTEMS. **Proceedings...** [S.l.: s.n.], 2014. p. 53–60.

MORGAN, F.; CAWLEY, S. Enhancing learning of digital systems using a remote FPGA lab. In: 6TH INTERNATIONAL WORKSHOP ON RECONFIGURABLE COMMUNICATION-CENTRIC SYSTEMS-ON-CHIP (RECOSOC). **Proceedings...** IEEE, 2011. p. 1–8. ISBN 978-1-4577-0640-0. Available from Internet: <https://ieeexplore.ieee.org/abstract/document/5981525http://ieeexplore.ieee.org/document/5981525/>.

MORGAN, F. et al. Remote FPGA Lab with Interactive Control and Visualisation Interface. In: 2011 21ST INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS. **Proceedings...** IEEE,

2011. p. 496–499. Available from Internet: <https://ieeexplore.ieee.org/abstract/document/6044871https://ieeexplore.ieee.org/document/6044871/>.

MOURSI, A. et al. Different reference models for uvm environment to speed up the verification time. In: 2018 19TH INTERNATIONAL WORKSHOP ON MICROPROCESSOR AND SOC TEST AND VERIFICATION (MTV). **Proceedings...** [S.l.: s.n.], 2018. p. 67–72.

NEDIC, Z.; MACHOTKA, J.; NAFALSKI, A. Remote laboratories versus virtual and real laboratories. **Proceedings - Frontiers in Education Conference, FIE**, Institute of Electrical and Electronics Engineers Inc., v. 1, p. T3E1–T3E6, 2003. Available from Internet: <https://ieeexplore.ieee.org/document/1263343>.

NEUSTOCK, L. T. et al. Scalable laboratory experimentation using ilabs - the digital twins for experiments. In: . [S.l.]: Optica Publishing Group, 2021. p. F2A.7.

NEUTZLING, A. et al. A simple and effective heuristic method for threshold logic identification. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 37, n. 5, p. 1023–1036, 2018.

NEUTZLING, A. et al. Synthesis of threshold logic gates to nanoelectronics. In: 2013 26TH SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI). **Proceedings...** [S.l.: s.n.], 2013. p. 1–6.

NEUTZLING, A. et al. Effective logic synthesis for threshold logic circuit design. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 38, n. 5, p. 926–937, 2019.

NEUTZLING, A. et al. Threshold logic synthesis based on cut pruning. In: 2015 IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN (ICCAD). **Proceedings...** [S.l.: s.n.], 2015. p. 494–499.

NICKELS, K. M. Pros and cons of replacing discrete logic with programmable logic in introductory digital logic courses. In: . ASEE Conferences, 2000. p. 5.511.1–5.511.7. Available from Internet: <http://peer.asee.org/8648>.

OpenCircuits. **OpenCircuits**. 2022. Available from Internet: <https://github.com/OpenCircuits/OpenCircuits/>.

ORACLE. **Beginner's Guide toOracle Grid Engine 6.2**. [S.l.], 2010. Accessed: 2022-12-04.

OURWORLDINDATA. **Moore's law: The number of transistors per microprocessor**. 2022. <https://ourworldindata.org/grapher/transistors-per-microprocessor>.

PADWICK, A. et al. Tackling the digital and engineering skills shortage: Understanding young people and their career aspirations. In: . [S.l.]: IEEE, 2020. p. 1–8.

PERALTA, R. D. et al. A method to join the on-set and off-set of an incompletely boolean function into a single bdd. In: 2021 34TH SBC/SBMICRO/IEEE/ACM SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI). **Proceedings...** [S.l.: s.n.], 2021. p. 1–6.

PERALTA, R. D. et al. An improved method to join bdds for incompletely specified boolean functions. In: 2023 IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS). **Proceedings...** [S.l.: s.n.], 2023. p. 1–5.

PFISTER, G. The yorktown simulation engine: Introduction. In: . [S.l.]: IEEE, 1982. p. 51–54. ISBN 0-89791-020-6.

POLI, R. et al. Unified theory to build cell-level transistor networks from bdds [logic synthesis]. In: 16TH SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 2003. SBCCI 2003. PROCEEDINGS.. **Proceedings...** [S.l.: s.n.], 2003. p. 199–204.

PULLI, A.; KREMASTIOTIS, I.; KULIS, S. Verification methodology of a multi-mode radiation-hard high-speed transceiver asic. **Journal of Instrumentation**, IOP Publishing, v. 17, n. 03, p. C03008, mar 2022.

RABAEY, J. M.; CHANDRAKASAN, A.; NIKLIć, B. **Digital Integrated Circuits: A Design Perspective**. [S.l.]: Pearson, 2003.

RAMAN, R. et al. Virtual laboratories- a historical review and bibliometric analysis of the past three decades. **Education and Information Technologies**, Springer, p. 1–33, 4 2022. ISSN 1360-2357. Available from Internet: <https://link.springer.com/article/10.1007/s10639-022-11058-9https://link.springer.com/10.1007/s10639-022-11058-9>.

RAMOS, J. A. G.; ALBERTINI, B.; SOLIS-LASTRA, J. A systematic literature review on laboratory as a service (laas). **2022 IEEE XXIX International Conference on Electronics, Electrical Engineering and Computing (INTERCON)**, IEEE, p. 1–4, 8 2022. Available from Internet: <https://ieeexplore.ieee.org/document/9870147/>.

RAVANASA, K.; HASHEMIAN, R. VLab, a high speed multi-accesses parallel processing remote laboratory access for FPGA design technology. **IEEE International Conference on Electro Information Technology**, IEEE Computer Society, p. 377–381, 2014. Available from Internet: <https://ieeexplore.ieee.org/abstract/document/6871794>.

REIS, A. Covering strategies for library free technology mapping. In: PROCEEDINGS. XII SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (CAT. NO.PR00387). **Proceedings...** [S.l.: s.n.], 1999. p. 180–183.

REIS, A.; DRECHSLER, R. **Advanced Logic Synthesis**. [S.l.]: Springer, 2018.

REIS, A.; MATOS, J. Physical awareness starting at technology-independent logic synthesis. In: REIS, A.; DRECHSLER, R. (Ed.). ADVANCED LOGIC SYNTHESIS. **Proceedings...** [S.l.]: Springer, 2018.

REIS, A. I. et al. Library free technology mapping. In: _____. VLSI: INTEGRATED SYSTEMS ON SILICON: IFIP TC10 WG10.5 INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION 26–30 AUGUST 1997, GRAMADO, RS, BRAZIL. **Proceedings...** Boston, MA: Springer

US, 1997. p. 303–314. ISBN 978-0-387-35311-1. Available from Internet: <https://doi.org/10.1007/978-0-387-35311-1_25>.

RODA-SEGARRA, J. Virtual laboratories during the covid-19 pandemic: A systematic review. In: . IEEE, 2021. p. 1–4. ISBN 978-1-6654-3703-5. Available from Internet: <https://ieeexplore.ieee.org/document/9600344/>.

RODRIGUEZ-GIL, L. et al. Graphic technologies for virtual, remote and hybrid laboratories: WebLab-FPGA hybrid lab. **Proceedings of 2014 11th International Conference on Remote Engineering and Virtual Instrumentation, REV 2014**, IEEE Computer Society, p. 163–166, 2014. Available from Internet: <https://ieeexplore.ieee.org/abstract/document/6784245>.

ROSA, L. S. et al. Scheduling policy costs on a java microcontroller. In: MEERSMAN, R.; TARI, Z. (Ed.). ON THE MOVE TO MEANINGFUL INTERNET SYSTEMS 2003: OTM 2003 WORKSHOPS. **Proceedings...** Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 520–533. ISBN 978-3-540-39962-9.

ROSA, L. S. da et al. A comparative study of cmos gates with minimum transistor stacks. In: PROCEEDINGS OF THE 20TH ANNUAL CONFERENCE ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN. **Proceedings...** New York, NY, USA: Association for Computing Machinery, 2007. (SBCCI '07), p. 93–98. ISBN 9781595938169. Available from Internet: <https://doi.org/10.1145/1284480.1284511>.

ROSA, L. S. da et al. Switch level optimization of digital cmos gate networks. In: 2009 10TH INTERNATIONAL SYMPOSIUM ON QUALITY ELECTRONIC DESIGN. **Proceedings...** [S.l.: s.n.], 2009. p. 324–329.

ROSENBERG, S.; MEADE, K. A. **A Practical Guide to Adopting the Universal Verification Methodology (UVM)**. [S.l.]: Cadence Design Systems, 2013.

ROSER, M.; RITCHIE, H.; MATHIEU, E. **Our World in Data**. 2023. <https://ourworldindata.org/technological-change>. Accessed: 2023-04-25.

SCHIRRMEISTER, F.; BERSHTEYN, M.; TURNER, R. Hardware-assisted verification and software development. In: _____. ELECTRONIC DESIGN AUTOMATION FOR IC IMPLEMENTATION, CIRCUIT DESIGN, AND PROCESS TECHNOLOGY. **Proceedings...** [S.l.]: CRC Press, 2017. p. 461–489.

SHAHDAD, M. et al. Vhsic hardware description language. **Computer**, v. 18, n. 2, p. 94–103, 1985.

SIEMENS. **Siemens partners with TSMC for 3nm product certifications and other technology milestones**. 2022. <https://www.plm.automation.siemens.com/global/en/our-story/newsroom/siemens-tsmc-oip-22/110386>. Accessed: 2022-11-26.

SIEMENS. **Veloce HW-Assisted Verification System**. 2022. <https://eda.sw.siemens.com/en-US/ic/veloce/>. Accessed: 2022-12-06.

SINGH, G. **Gate-Level Simulation Methodology**. [S.l.], 2015.

SKYWATER. **SkyWater Foundry Provided Standard Cell Libraries**. 2022. <https://skywater-pdk.readthedocs.io/en/main/index.html>. Accessed: 2022-11-24.

SMIL, V. July 1958: Kilby conceives the ic. **IEEE Spectrum**, Institute of Electrical and Electronics Engineers Inc., v. 55, p. 22–22, 7 2018. ISSN 0018-9235. Available from Internet: <https://ieeexplore.ieee.org/document/8389182/>.

Stefan Belinov. **Logic Circuit Simulator Pro**. 2022. Available from Internet: <https://play.google.com/store/apps/details?id=com.duracodefactory.logiccircuitsimulatorpro&hl=en_US&gl=US>.

Stephen Williams. **Icarus Verilog**. 2022. Available from Internet: <http://iverilog.icarus.com/>.

Steven Kollmansberger. **Logic Gate Simulator**. 2022. Available from Internet: <https://www.kolls.net/gatesim/>.

SYNOPSYS. **Emulation**. 2022. <https://www.synopsys.com/verification/emulation.html>. Accessed: 2022-12-06.

SYNOPSYS. **Technology Access Program (TAP-in)**. 2022. <https://www.synopsys.com/community/interoperability-programs/tap-in.html>. Accessed: 2022-11-29.

SYNOPSYS. **TSMC & Synopsys: Collaborate to Innovate**. 2022. <https://www.synopsys.com/partners/tsmc.html>. Accessed: 2022-11-26.

TIMUR, L.; XIE, Y. Is border closure effective in containing covid-19? **Travel Medicine and Infectious Disease**, Elsevier, v. 44, p. 102137, 11 2021. ISSN 14778939. Available from Internet: <https://linkinghub.elsevier.com/retrieve/pii/S1477893921001782>.

TOGNI, J. et al. Automatic generation of digital cell libraries. In: PROCEEDINGS. 15TH SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN. **Proceedings...** [S.l.: s.n.], 2002. p. 265–270.

TRIMBERGER, S. M. Three ages of fpgas: A retrospective on the first thirty years of fpga technology. **Proceedings of the IEEE**, Institute of Electrical and Electronics Engineers Inc., v. 103, p. 318–331, 3 2015. ISSN 0018-9219. Available from Internet: <http://ieeexplore.ieee.org/document/7086413/>.

VICILOGIC. **viciLogic.com**. 2022. Available from Internet: <https://www.vicilogic.com/>.

VILLAR-MARTINEZ, A. et al. Towards Reliable Remote Laboratory Experiences: A Model for Maximizing Availability Through Fault-Detection and Replication. **IEEE Access**, Institute of Electrical and Electronics Engineers Inc., v. 9, 2021. Available from Internet: <https://ieeexplore.ieee.org/document/9376937https://ieeexplore.ieee.org/document/9376937/>.

VILLAR-MARTINEZ, A. et al. Toward widespread remote laboratories: Evaluating the effectiveness of a replication-based architecture for real-world multiinstitutional usage. **IEEE Access**, Institute of Electrical and Electronics Engineers Inc., v. 10, p. 86298–86317, 2022. ISSN 2169-3536. Available from Internet: <https://ieeexplore.ieee.org/document/9857860/>.

VILLAR-MARTINEZ, A. et al. Improving the Scalability and Replicability of Embedded Systems Remote Laboratories Through a Cost-Effective Architecture. **IEEE Access**, Institute of Electrical and Electronics Engineers Inc., v. 7, p. 164164–164185, 2019. Available from Internet: <https://ieeexplore.ieee.org/document/8894124https://ieeexplore.ieee.org/document/8894124/>.

Von Braun Labs. **#Data**. 2022. Available from Internet: <http://design.vonbraunlabs.com.br/>.

WAGNER, F. R.; REIS, A. I.; RIBAS, R. P. **Fundamentos de circuitos digitais**. [S.l.]: Sagra Luzzatto, Porto Alegre, 2006.

WANG, L.-T.; CHANG, Y.-W.; CHENG, K.-T. T. **Electronic design automation: synthesis, verification, and test**. [S.l.]: Morgan Kaufmann, 2009.

WESTE, N.; HARRIS, D. **CMOS VLSI Design: A Circuits and Systems Perspective**. 4th. ed. USA: Addison-Wesley Publishing Company, 2010. ISBN 0321547748.

WIKIPEDIA. **Intel 8086**. 2023. <https://en.wikipedia.org/wiki/Intel_8086>.

WIKIPEDIA. **MOS Technology 6502**. 2023. <https://en.wikipedia.org/wiki/MOS_Technology_6502>.

WIKIPEDIA. **Ricoh 5A22**. 2023. <https://en.wikipedia.org/wiki/Ricoh_5A22>.

WIKIPEDIA. **Transistor Counter**. 2023. <https://en.wikipedia.org/wiki/Transistor_count>.

WIKIPEDIA. **Zilog Z80**. 2023. <https://en.wikipedia.org/wiki/Zilog_Z80>.

XILINX. **7 Series FPGAs Configurable Logic Block, User Guide, v1.8**. [S.l.], 2016.

ZAMAN, M. A.; NEUSTOCK, L. T.; HESSELINK, L. Ilabs as an online laboratory platform: A case study at stanford university during the covid-19 pandemic. **IEEE Global Engineering Education Conference, EDUCON**, IEEE Computer Society, v. 2021-April, p. 1615–1623, 4 2021. ISSN 21659567.

ZHANG, Y.; REN, H.; KHAILANY, B. Opportunities for rtl and gate level simulation using gpus. In: PROCEEDINGS OF THE 39TH INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN. **Proceedings...** [S.l.]: Association for Computing Machinery, 2020.